Lecture Notes in Artificial Intelligence 3825

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Michael G. Hinchey Patricia Rago
James L. Rash Christopher A. Rouff
Roy Sterritt Walt Truszkowski (Eds.)

Innovative Concepts for Autonomic and Agent-Based Systems

Second International Workshop
on Radical Agent Concepts, WRAC 2005
Greenbelt, MD, USA, September 20-22, 2005
Revised Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Michael G. Hinchey
James L. Rash
Walt Truszkowski

NASA Software Engineering Laboratory, NASA Goddard Space Flight Center
Greenbelt, MD 20771, USA
E-mail: {Michael.G.Hinchey,James.L.Rash,Walter.F.Truszkowski}@nasa.gov

Patricia Rago
IBM Autonomic Computing
Raleigh, NC, USA
E-mail: ragopat@us.ibm.com

Christopher A. Rouff
SAIC
McLean, VA 22102, USA
E-mail: rouffc@saic.com

Roy Sterritt
University of Ulster
Jordanstown, Northern Ireland
E-mail: r.sterritt@ulster.ac.uk

Library of Congress Control Number: 2006938753

CR Subject Classification (1998): I.2.11, I.2, D.2, C.2.4, H.5.2-3

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-69265-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-69265-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11964995 06/3142 5 4 3 2 1 0

Preface

The second WRAC, NASA/IEEE Workshop on Radical Agent Concepts, was held at NASA Goddard Space Flight Center, Greenbelt, MD, September 20–22, 2005.

The workshop was sponsored by the Information Systems Division of NASA Goddard and IEEE Computer Society Technical Committee on Complexity in Computing and IEEE Task Force on Autonomous and Autonomic Systems. The workshop also received generous financial support from IBM, without which the workshop would not have been possible.

Agent technology, along with autonomous and autonomic computing, has emerged as a major field in computing, and will greatly influence the future development of complex computer-based systems. The area of research is strongly influenced by the autonomic computing initiative as well as by developments in biologically inspired computing, and involves interdisciplinary interaction from those involved in research in social intelligence, psychology, arts, biology, computer science, computer communications and philosophy.

This volume includes revised versions of papers presented at the workshop. The workshop was structured so as to allow adequate time for discussion and interaction, to exchange ideas and reflect on the motivations, scientific grounds and practical consequences of the concepts presented. Many of the ideas are truly "radical", and so authors were given time to revise their papers to reflect further thoughts on the ideas presented and to reflect feedback received at the workshop.

We are grateful to Jeff Kephart for a very interesting keynote speech describing IBM's current and future work in this field, which fit very well with the aims and scope of the workshop.

Again, we would like to express our thanks to NASA Goddard, IEEE Computer Society and IBM for their support of this event. Our thanks also to Alfred Hofmann and Anna Kramer of Springer for their support and interest in publishing these proceedings. We hope the papers are of interest to the reader, and we hope to welcome you soon to the next Workshop on Radical Agent Concepts.

October 2006

Mike Hinchey
Patricia Rago
Jim Rash
Chris Rouff
Roy Sterritt
Walt Truszkowski

Table of Contents

Agent-Mediated Pro-active Web-Sites

Karin Breitman[1] and Walt Truszkowski[2]

[1] Pontificia Universidade do Rio de Janeiro
karin@inf.puc-rio.br
[2] NASA Goddard Space Flight Center
Walt.Truszkowski@nasa.gov

Abstract. In this paper we explore the concept of a Semantic Desktop in which software applications will be responsible for the automation of otherwise tedious tasks and the update of user information. We propose an ontology driven software agent solution where two major issues stand out: architecture and process. The architecture issue addresses the current static nature of information on a web page. In order to achieve the level of pro-activity desired we have to be able to identify, classify and code important bits of semantic information in a way that can be automatically manipulated by machines. The second issue relates to the strategy/strategies used to dynamically update information. Because of the variety of actions needed to update web sites e.g., monitoring changes to personal information (phone numbers, addresses, hobbies, etc.), maintaining a knowledge of institutional information, reacting to external events, reaction to political and environmental changes, etc., we hypothesize that no single strategy for maintaining dynamic information-awareness will do but that a variety of strategies need to be put into practice. We propose the creation of a community of software agents, each responsible for achieving a different set of tasks. The Semantic Desktop is achieved through the composition of the site semantics with the functionality provided by the instantiated software agents.

1 Introduction

A few years ago, we were all impressed by the statistics that over a million VCRs were constantly blinking "00:00" because their users were either ignorant about how to correct the situation or did not care to update the time. The same situation seems to be happening to personal web sites. Owners, too busy to update information, sometimes leave their pages unattended for years. Nothing is more frustrating than finding an interesting web-site that is not current A major contributing reason for out-of-date web-sites is rooted in the very nature of the web itself. Information changes so rapidly that most users feel that by the time they make needed adjustments to their web-site, it is already becoming obsolete. In this paper we argue that it is possible to semi automate the information-update process with the use of emergent software-agent technology [Williams04, Pazzani02, Woolridge99, Sycara98].

The basic idea, explored and expounded upon in this paper, is to make web-sites more pro-active in providing user centered services and updating their posted

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 1 – 12, 2006.

information. In order to do this, there needs to be access to web-based information in a format that allows for automatic manipulation of the data. The exponential growth of the Internet has dramatically increased the number of available electronic resources, e.g. articles, books, web pages and data repositories, making it more difficult to find, manipulate and present information without the aid of automation [Fensel02].

Researchers worldwide are investigating the possibility of creating a Semantic Web, an improvement to today's web, in which resources will be given meaning through machine processable/parseable languages [Berners-Lee01]. As a result, a gamut of new technologies has emerged. Among others there is the use of metadata, ontologies, semantic web services, semantic navigation, and software agents. We will explore these technologies as they pertain to the novel approach to the development of self adapting web sites that we are proposing.

2 Metadata

Most of the information available in the web today is formatted for user consumption, i.e, machines play a very small role in the interpretation and processing of information. Most data is captured using markup languages that rely on the use of tags or labels that have meaning to human beings alone. Other than processing some interfaces cues, such as font, color and heading placements, computers are able to process a very small share of the available information in the nearing 8 billion web pages.

In order to allow computers to take up a larger share of the processing, we have to provide information in a format that allows for automatic manipulation. Researchers worldwide are investigating the possibility of creating a Semantic Web, an improvement to today's web in which resources will be given meaning through machine processable languages [Berners-Lee01, Antoniou04, Gómez-Pérez04]. As a result a gamut of new technologies have emerged. Among others there is the use of metadata, ontologies, semantic web services, semantic navigation, and software agents. We will explore some of these technologies as to provide a novel approach to the development of the proposed Scientific Desktop.

In our understanding, the information relevant to the Scientific Desktop is a composition of hybrid elements, text, image, sound or video content. The set of elements are typically interrelated and organized in some coherent, however domain dependent fashion, e.g., in a personal web page we expect to find the information of an individual in the top of a page whereas at product page we would expect its specifications to come before the manufacturer's contact data.

A first step in the preparation of the information for automated processing is identifying a Taxonomy of information types, e.g. research area, technology used, contact information, institutional policies. That is not a trivial task, for the relevant information is presented in varying degrees of abstraction, for example:

- **High level (general)** – what the person is: researcher, vendor, businessman…
- **Medium level** – if researcher, what area: types of events, technology (if any), association to laboratories, entities (NFS, for instance), institutions, standards and rules that it must abide to....

- **Low level (specific)** – specific interests, colleagues it works with, institutions is connected to, special interest groups and associations he or she belongs to…

Once we identify the information type, it is possible to associate an adequate Metadata, data about data, representation for the information in question [Gilliland-Sweland97]. In the Research & Academic context most types of information are already organized by some existing taxonomy, such as the list of the Computer Science sub areas by IEEE or the NAICS. Those have to be "re-written" in some metadata compatible language, RDF for instance. Other examples are:

- People – Friend of a Friend,
- Publications – Dublin Core,
- Projects – RDF Taxonomy structured as the NFS list of research areas,
- Areas – RDF Taxonomy structured as the Societies lists of topics (for CS we could use IEEE's Computing Society),
- Physical Location and Contact information – RDF version of the University Directory, Floor Plan, vCard taxonomy,
- Academic Undergrad/Graduate Courses – RDF version of the University Catalogue.

We exemplify those information chunks using the example of the home page of a researcher, depicted in Figure 1. Bellow we list a series of information chunks that represent the information that is available on the page.

Information available:
Bio, Bibliographical Production (Conference Proceedings, Tech Reports, Books, chapters), People (Colleagues, Students, Fellow Research Project Members), Research areas(Requirements Engineering, Semantic Web, Formal Systems) Events and Conferences she is organizing, Consulting, Activity in Scientific Societies.

Fig. 1. Researcher's Home Page

For each item present in the page some annotation describing its provenance, ownership and categorization is required. To exemplify we introduce one of the many links to bibliographical references, in this case an article in a Conference:

[Breitman04] - Breitman, K.K.; Leite, J.C.S.P - Lexicon Based Ontology Construction - Lecture Notes in Computer Science 2940- Editors: Carlos Lucena, Alessandro Garcia, Alexander Romanovsky, et al. - ISBN: 3-540-21182-9 - Springer-Verlag, February 2004, pp.19-34.

Possibly the best way to describe this item would be using the Dublin Core Metadata Schema [Gill97, DC05], as illustrated in Figure 2.

Dublin Core

1. Subject: Software Engineering ***(list of subjects in IEEE)***
2. Title: Lexicon Based Ontology Construction
3. Creator : Karin Breitman
4. Description: article
5. Publisher: Springer Verlag
6. Contributor) : Julio Cesar Sampaio do Prado Leite
7. Date: February 2004
8. Type: Scientific Article
9. Format: text
10. Identifier : ISBN: 3-540-21182-9
11. Relation: ***ESSMA Project*** , ***C&L Tool***
12. Source:
13. Language: English
14. Coverage:
15. Rights: Springer Verlag

Fig. 2. Dublin Core Metadata Annotation for a bibliographical reference

In this case there is a hierarchy of other information nested within this reference. The co-author of the article, a colleague named Julio Cesar Sampaio do Prado Leite is an piece of information its own right and needs to be unambiguously reference. A possible metadata representation for it would use the FOAF (friend of a friend RDF markup), as follows:

```
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/">
<foaf:Person>
      <foaf:name >Julio Cesar Sampaio do Prado Leite</foaf:name>
      <foaf:mbox rdf:resource="mailto: julio@inf.puc-rio.br " />
      <foaf:nick >Julio</foaf:nick>
      <foaf:workplacehomepage rdf:resource="http:/www.inf.puc-rio.br/~julio" />
</foaf:Person>
   </rdf:RDF>
```

Fig. 3. Friend of a Friend(FOAF) RDF representation of a colleague (co-author)

The metadata markup will be very useful in referencing and storing pieces of information. However its semantics is still very weak. In addition to the information provided about the information objects themselves, we also want to represent properties, rules and relationships that are hold among information chunks. For that purpose we propose to adopt the use and integration of ontologies. Ontologies provide a shared understanding of a given domain by providing formal definition to its main concepts, properties and relationships that hold among those. The ontology

construction process is inherently compositional, in that a new domain ontology is usually build by reusing parts of existing ontologies. In the case of the proposed Scientific Desktop, there are available ontologies that can be reused in the process [Gómez-Pérez98, Guarino02].

3 Ontology

The metadata markup used to index the researcher's example in the last section is very useful in referencing and storing pieces of information, but its semantics is still very weak for our intended purposes. In addition to the information provided about the object itself, metadata, we also want to represent properties, truths (axioms) and relationships that are hold among information chunks. For that purpose we adopt the ontological representation. An ontology, as defined by Alexander Maedche [Maedche02] is described by a 5-tuple containing its core elements, i.e., concepts, relations, hierarchy, a function that relates concepts non-taxonomically and a set of axioms. The elements are defined as follows:

$O := \{C, \mathcal{R}, \mathcal{HC}, rel, \mathcal{A}_O\}$ consisting of :

- Two disjoint sets, C (concepts) and $\mathcal{R}$ (relations),
- A concept hierarchy, $\mathcal{H}_C$: $\mathcal{H}_C$ is a directed relation $\mathcal{H}_C \subseteq C \times C$ which is called concept hierarchy or taxonomy. $\mathcal{H}_C(C1,C2)$ means $C1$ is a subconcept of $C2$,
- A function $rel: \mathcal{R} \rightarrow C \times C$ that relates the concepts non taxonomically,
- A set of ontology axioms A_O, expressed in appropriate logical language.

Figure 4 illustrates a simple example of this ontological structure. We borrow from the Researcher's Home Page for this example. We depict three concepts, Project(x), Long_term_project(y) and Researcher(z), one relation, participates_in(w) that holds between concept Long_tem_project and Researcher. The concept

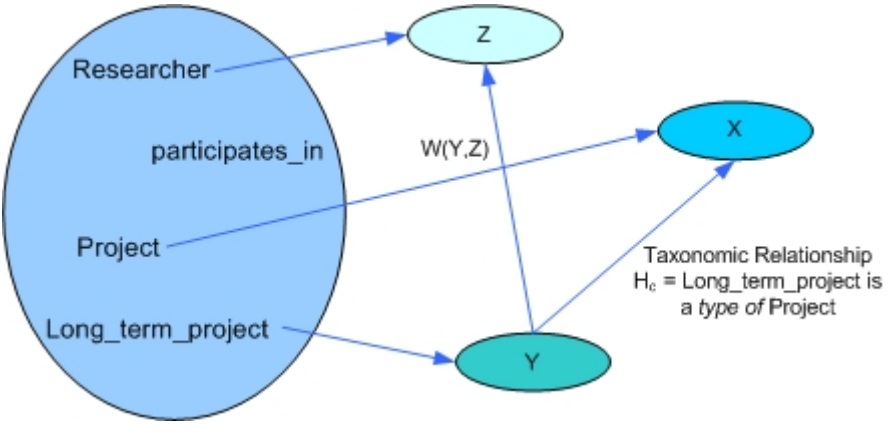


Fig. 4. Example of ontological elements and their relations using the ontology structure O proposed by [Maedche02]

Long_term_project is a subconcept of Project. According to the structure we have something like, C : = {x, y, z} and R : = {w}. The following hierarchical relation, H_C (y,x) and the non taxonomic relation, w(y,z).

Ontologies provide a shared understanding of a given domain by providing formal definition to its main concepts, properties and relationships that hold among those [Gruber93, Gómez-Pérez04, Fensel03, Breitman05]**.** The ontology construction process is inherently compositional, in that a new domain ontology is usually build by reusing parts of existing ontologies. In the case of the Researcher Home Page example, we can easily identify different sources that could serve in composing the information chunks concept sub trees. In table 1we exemplify some possibilities.

Table 1. Reusable Information sources for ontological composition

Information Chunk Type	**Reusable Information Sources**
People	FOAF Taxonomy of family and work relationships,
Research Area	IEEE taxonomy, NAICS
Events	University ontology, IEEE/ACM calendar of events
Projects	Subcategories of NSF and/or Esprit European projects
Physical Location and Contact information	University Directory, Floor Plan, vCard taxonomy
Courses	University Catalogue

In practice most of the top nodes of the ontology, i.e., the ones that represent more general concepts, can be reused from existing ontologies. As we navigate down, specializing the concepts, we identify the need of incorporating information specific to that particular ontology. Most of the construction effort will be spent in the identification and incorporation of specialized (leaf) concepts and their instances to the ontology.

In figure 5 we show part of the Researcher Home Page ontology. We can identify three zones. The top most levels are concepts so general that could be used in the creation of any personal home page. These concepts can be reused in the spirit of upper ontologies, such as Cyc [OpenCyc] and WordNet [Miller95], that embody consensual knowledge.

The second zone corresponds to the specialization to the computer science area. We would like to call this group level specialization (university researcher in the computer science area). At this level reuse is viable, but requires some adaptation level.

At the third and bottommost level, the concepts and instances added to the ontology reflect particular information about the individual that own this particular page. At this level very little reuse can be accomplished. Once ready, this ontology will model the preferences of a single, or at most a reduced number of users. Based on previous collaboration in the area of Human Computer Interaction, we are currently

investigating the possibilities of using ontologies as user models [Barbosa04, Pazzani97, Perkowitz04].

This feasibility of this approach is very important to this research, because it will shape the architecture of the software agents we want to implement. If we are able to concentrate the user profile in his or her ontology, the construction and operation of the software agents will be more general and thus increase the level of agent reuse [Holvoet03].

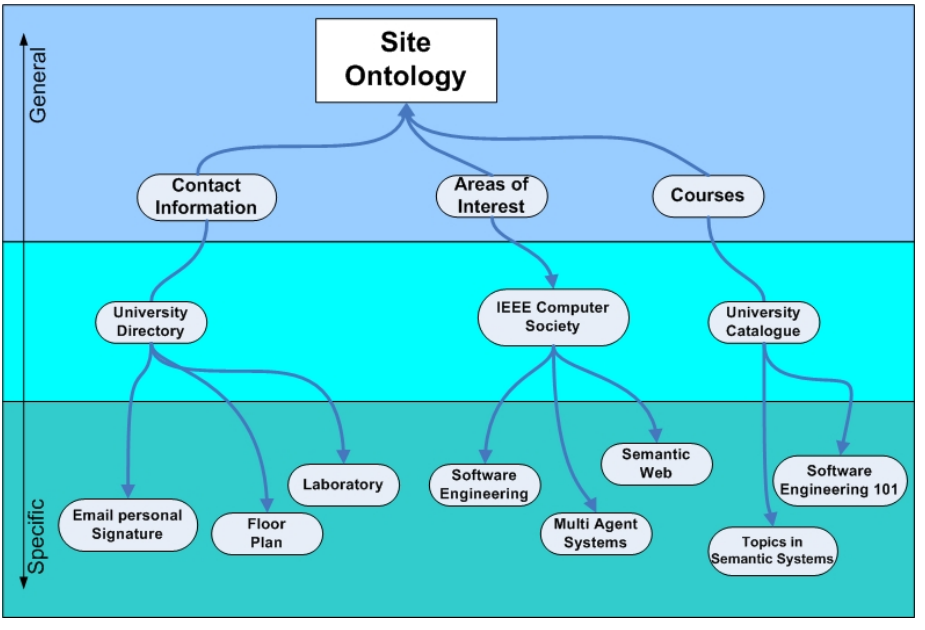


Fig. 5. Researcher Ontology (partial)

In the next section we discuss the requirements and propose an architecture, based in the concept of ontology driven software agents, to implement the proposed adapting web sites.

4 Adaptive Web-Sites

In the previous section we discuss the organization that has to imposed to the information resources in order to allow for automated computer processing. The nature of the information suggests that there are two possible types of sources:

- Internal (privately owned) – information that comes from the owner's own computer (e-mail, recent files, articles recently written/submitted, discussion groups, directory structure itself...)

- External (worldly) – from syndication (web sites) and from a myriad of heterogeneous, distributed sources (conferences, publishers, Amazon, scientific societies, colleague home pages...)

The current exponential growth of web accessible resources has created a demand for new tools to aid people cope with the volume of information. Examples of such tools are indexing mechanisms, used in the organization of large data repositories, such as the NAICS and IEEE catalogues and Yahoo's index of subjects. Balabanovic et al propose a system in a software agent learns users patterns and browses the Internet in his or her behalf [Balabanovic95]. Etzioni investigates the possibilities of Semantic email [Etzioni04], Rucker and Polancon made CiteSeer available, a site that uses references in scientific articles that serves as guide to other works of interest [Rucker97]. There is a myriad of information sources available in today's web, raging from static elements (written text in pages and files) to the results of behavioral patterns such as navigation and user preferences. It is clear that different information mining strategies will be needed, case by case. We propose to incorporate such strategies in the profiles of different software agents [Jennings98, Woolridge95, Hendler01]. The term agent, as defined by Woolridge and Jennings denotes software based computer systems that enjoy the properties of : autonomy, social ability, reactivity and pro-activeness [Woolridge95].

Agent based architectures became an alternative that helps deal with complexity and increasing volumes of information. Multi agent systems are being used to build information and knowledge sharing systems [Dignum03, Abecker03, Chen00] The MACRON architecture, proposed by Decker et al, was designed specifically for the proactive acquisition of information in complex information gathering environments [Decker05]. This architecture relies in the use of agents to find and conciliate heterogeneous information from different sources. Agent based systems are also being used to build organization information systems, such as the FRODO project, whose goal is to build distributed organizational memory systems. [Abecker03].

The architecture of our solution will be a composition of diverse software agents, i.e., a community of agents or multi agent system (MAS). Ferber defines a multi agent system as a "collection of possibly heterogeneous, computational entities, having their own problem solving capabilities and which are able to interact among them in order to reach an overall goal" [Ferber99]. Among the most distinctive characteristics of Multi agent systems are the decentralization of data, lack of global control and the fact that each individual agent has a partial view (and capabilities of solving) the problem [Jennings98].

Initially we envision the creation of four agents:

4.1 External Agents

Syndicating Agent

This agent will roam the web cataloguing information of interest to the site . Can be implemented using the RSS - Really Simple Syndication format – also called “feed”(XML format that allows for the update of information links on other sites). Growing number of adept sites (New York Time, BBC and MIT Press, among

others). The downside is that RSS is a XML dialect, so it provides no semantics. We propose to overcome this effect by combining the use of RSS markup with ontology and metadata.

Serendipity Agent
Besides syndicating known well known pages, we propose a serendipity agent, whose goal is to roam the web in search of new, interesting pieces of information. The concepts it will be searching are those in the site ontology. The agent should identify important concepts and aligning those to pages in the Internet in a serendipity manner. Strategies can be:

–Similarity,
–"*If you like this ...*",
–Semantic Search (ontology driven) using Google & Google Scholar.

4.2 Internal Agents

Indexing Agent
This agent will be responsible to update the site ontology based on the information obtained when indexing the site owner's computer(s). It can work in conjunction with indexing software, such as the Google Desktop Search Plug-in or Copernic desktop search. This agent will provide the information needed to update the most specialized links in the site ontology, files and at the Outlook address book. Because it indexes personal files, e-mails, and e-mail attachments stored anywhere on the owners PC, the information it contains reflects the owner's personal interests and choices.

Spy Agent
The role of this agent is to monitor the site owner's activities. The behavior of the site owner can tell much about his or her preferences, among others:

- Internet – sites he or she visits frequently, list of preferred sites, pop ups he or she always dismisses.
- Mail – monitoring the e-mail can help identify, among others:
    - Colleagues the owner corresponds frequently to,
    - Confirmation on ticket and hotel reservations (update Outlook),
    - Notification of acceptance of papers,
    - News that appeared on lists he or she subscribes to.

5 Conclusion

In this paper we argument in favor of an ontology driven, agent-mediated, adaptive web sites. Our ideas, different from the argumentation used in the customization of web sites based on user group navigational patterns [Pazzani97], is to create a single, personalized web environment that serves its owner. Among others it will automate tedious and repetitive tasks, such as the update of CVs, agenda, contact information, trip related information (directions, hotel names and locations, airplane ticketing information), present automatic update/alerts on what colleagues/partners are doing at

present (articles published, books in print, presentation they gave..), automatically change web pages and include new links of interest and provide an understandable model of the user profile (user ontology).

We propose a flexible, compositional, customizable and extensible architecture that allows the site owner to decide what kind of services he or she would like to automate. The services provided by the software agents are mutually independent and can be combined as needed. New software agents can be included in the architecture providing additional services in the future.

Future work includes the visualization and navigation of data in the Semantic Desktop context. The same information chunk, the name of a fellow researcher for example, may have different implications depending on the context it is encountered. In Figure 6 we exemplify one such situation. At the center of the figure we depict one information chunk, namely a researcher named Alan M. Davis. This researcher is, at the same time, book author, active participant in the Requirements Engineering Research community, participated at event X and belongs to the e-mail address book.

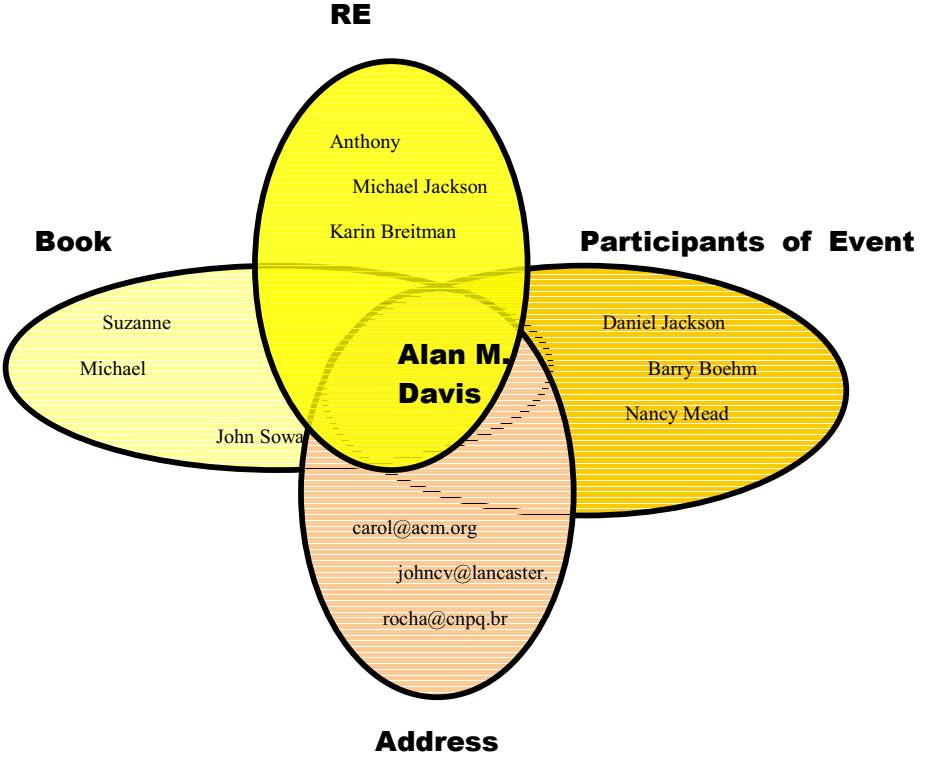


Fig. 6. Different Contexts for the IC "*Alan M. Davis*"

This phenomena is known as Polisemy in Natural Language Processing, where the same word or concept may assume more than one meaning, depending on the context of use. Dictionaries, such as the WordNet, catalog the possible meanings, also known as senses, that a given concept may assume [Miller95]. Its correct interpretation depends on the concept itself and some understanding of the context it is being used. Similarly, in the case of the Semantic Desktop, some indication of context may be necessary, in addition to the metadata associated to the concepts in question. We are currently investigating the adoption of the SHDM (Semantic Hypermedia Design Method) [Schwabe04]. In SHDM there is an explicit separation between the conceptual from the navigational design of Semantic Web Applications. Our proposal already addresses the conceptual design of the Semantic Desktop, with the use of Ontology and Metadata. The navigation design, on the other hand, would be responsible for the organization of the information in different contexts, allowing users of the Semantic Desktop to visualize and navigate his or her information space in a relevant manner.

References

[Abecker03] Abecker, A. ; Bernardi, A.; van Elst. - Agent technology for distributed organizational memories. In Proceedings of the 5th International Conference On Enterprise Information Systems, Vol. 2, 2003. pp. 3–10

[Antoniou04] Antoniou, G.; Harmelen, F.; - A Semantic Web Primer – MIT Press, Cambridge Massachussets, 2004.

[Balanovic95] Balabanovic, M, Shoham Y., - Learning Information Retrieval Agents: Experiments with Automated Web Browsing - Proceedings of the {AAAI} Spring Symposium on Information Gathering from Heterogenous, Distributed Resources, 1995. pp. 13--18.

[Bradshaw95] Bradshaw, J. M.; Dutfield, S. ; Carpenter, B. ; Jeffers, R.; Robinson, T. - KAoS: A generic agent architecture for aerospace applications. In T. Finin and J. Mayfield, editors, Proceedings of the CIKM ’95 Workshop on Intelligent Information Agents, Baltimore, Maryland, 1995.

[Breitman05] Breitman, K.K.; Haendchen, A.; von Staa, A.; Haeusler, H. - Ontologies to Formalize Services Specifications in Multi-Agent Systems. In: Lecture Notes in Artificial Intelligence. . Formal Approaches to Agent-Based Systems: International Workshop, FAABS. Springer Verlag, Heidelberg LNAI 3228-0092, 2005. – pp 92-110.

[Chen00] Chen, J.; Wolf, S.; Wragg, S. - A distributed multi-agent system for collaborative information management and sharing. In Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM). ACM Press, New York, NY, USA, 2000. pp. 382–388

[DC05] Páginas do Dublin Core – http://purl.org/DC/

[Decker95] Decker, K. ; Lesser, V. ; Nagendra Prasad, M.V. ; Wagner, T.- MACRON: An Architecture for Multi-agent Cooperative Information Gathering. In Proceedings of the CIKM '95 Workshop on Intelligent Information Agents. 1995

[Dignum03] Dignum, V.- Using Agent Societies to Support Knowledge Sharing. In AAMAS-03Workshop on Autonomy, Delegation and Control, Melbourne, Australia, July 14th 2003.

[Etzioni04] Etzioni, O. Et al - Semantic Email - Proceedings of the Thirteenth International World Wide Web Conference (WWW'04), 2004.

[Fensel02] Fensel, D. - Ontology Based Knowledge Management - IEEE Computer - November 2002. pp. 56-59

[Fensel03] Fensel, D.; Hendles, J.; Lieberman, H.; Wahlster, W.; Spinning the Semantic Web: Bringing the World Wide Web to its full potential editado por: Dieter Fensel, James Hendler, Henry Lieberman e Wolfgang Wahlster – MIT Press - 2003 - pp.1-25

[Ferber99] Ferber, J. - Multi-Agent System: An Introduction to Distributed Artificial Intelligence. Harlow: Addison Wesley Longman. 1999

[Gill97] Gill, T. – Metadata and the World Wide Web - em Metadata: Pathways to Digital Information – edited by Murtha Baca – Getty Information Institute, 1997. pp.9-18.

[Gilliland-Sweland97] Defining Metadata – em Metadata: Pathways to Digital Information – edited by Murtha Baca – Getty Information Institute, 1997. pp.1-8.

[Gómez-Pérez04] Gómez-Pérez, A.; Fernadéz-Peréz, M.; Corcho, O. - Ontological Engineering - Springer Verlag - 2004.

[Gómez-Pérez98] Gómez-Pérez, A. – Knowledge sharing and reuse – in The Handbook of Applied Expert Systems. CRC Press, 1998

[Gruber93] Gruber, T.R. – A translation approach to portable ontology specifications – Knowledge Acquisition – 5: 199-220

[Guarino02] Guarino, N.; Welty, C. – Evaluating Ontological Decisions with Ontoclean CACM Vol.45 No.2 – 2002. pp 61-65

[Hendler01] Hendler, J. – Agents and the Semantic Web – IEEE Intelligent Systems – March/April - 2001. pp.30-37

[Holvoet03] Holvoet, T., Steegmans, E. - Application Specific Reuse of Agent Roles - Lecture Notes in Computer Science 2603- Editors: Alessandro Garcia, Carlos Lucena,et al. - ISBN: 3-540-21182-9 - Springer-Verlag Berlin, Heidelberg, 2003, pp.148-164.

[Jennings98] Jennings, N.R.; Sycara, K. - Woolridge, M. - A Roadmap for Agent Research and Development. In Autonomous and Multi-Agent Systems, 1, 1998, pp. 7-38

[Miller95] Miller, G. – WordNet: A Lexical Database for English – Communications of the ACM – November 1995 pp.39-41

[Miller95] Miller, G. – WordNet: A Lexical Database for English – Communications of the ACM – November 1995 pp.39-41

[Open Cyc] http://www.opencyc.org/

[Pazzani02] Pazzani, M. Billsus, D - Adaptive Web Site Agents - Autonomous Agents and Multi Agent Systems 5, 2002. pp. 205-218.

[Pazzani97] Michael Pazzani, Daniel Billsus - Learning and Revising User Profiles: The Identification of Interesting Web Sites - Machine Learning Journal - Vol 27 Number 3, 1997. pp. 313-331.

[Perkowitz04] M. Perkowitz, M. Philipose, K. Fishkin, and D. J. Patterson. "Mining Models of Human Activities from the Web." Proceedings of the Thirteenth International World Wide Web Conference, 2004.

[Rucker97] Rucker J., Polanco M.J. - *SiteSeer: personalized navigation for the Web*, Communications of the ACM, Vol 40 Number 3, 1997

[Schwabe04] Design and Implementation of Semantic Web Applications. - Proceedings of the WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, New York, NY,USA, May 18, 2004

[Sycara98] Sycara, K. - Multiagent Systems - IA Magazine Vol 19 Number 2, 1998. pp. 79-92.

[Williams04] Williams, A.B.: Learning to Share Meaning in a Multi-Agent System. Journal of Autonomous Agents and Multi-Agent Systems, Vol. 8, No. 2, 165-193, March 2004.

[Wooldridge95] Wooldridge, M.; Jennings, N.; Intelligent agents: theory and practice. Em: Knowledge Engineering Review, v. 10, n. 2, p. 115-152; 1995.

[Wooldrige99] M. Wooldridge, N. R. Jennings, and D. Kinny: A methodology for agent-oriented analysis and design. In O. Etzioni, J. P. Muller, and J. Bradshaw, editors, Agents '99: Proceedings of the Third International Conference on Autonomous Agents, Seattle, WA, May 1999.

Learning to Use Referrals to Select Satisficing Service Providers

Teddy Candale and Sandip Sen

Mathematical & Computer Sciences Department
University of Tulsa
{teddy-candale,sandip}@utulsa.edu

Abstract. We investigate a formal framework where agents use referrals from other agents to locate high-quality service providers. Agents have common knowledge about providers which are able to provide these services. The performance of providers is measured by the satisfaction obtained by agents from using their services. Provider performance varies with their current load. We assume that agents are truthful in reporting interaction experiences with providers and refer the highest quality provider known for a given task. The referral mechanism is based of the exchange value theory. Agents exchange both the name of the provider to use and the satisfaction obtained by using a referred provider. We present an algorithm for selecting a service provider for a given task which includes mechanisms for deciding when and who to ask for a referral. This mechanism requires learning, over interactions, both the performance levels of different service providers, as well as the quality of referrals provided by other agents. We use a satisficing rather than an optimizing framework, where agents are content to receive service quality above a threshold. We experimentally demonstrate the effectiveness of our algorithm in producing stable system configurations where reasonable satisfaction expectations of all agents are met.

1 Introduction

Agents searching for high-quality service providers can either rely on their own interaction experience or use referral services, or referrals from other agents. We assume that the performance of a provider depends both on its intrinsic characteristics and the current workload it is handling. There can also be some intrinsic variabilities in the performance of a provider for the same given workload. A key question to consider in referral systems is the motivation for referring a resource or a provider to another agent. Agents need to find the best tradeoff between helping others with good referrals where the expectation is to get quality referrals in exchange and its potential long-term loss due to increased usage of the referred provider by the requesting agent.

While ideally speaking agents may aspire for optimal satisfaction levels from service providers selected for performing an assigned task, dynamic, partially known, and open environments can render the realization of this ideal behavior

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 13–22, 2006.

improbable. Possible sources of inefficiencies include noisy, variable feedback about provider performance as the environment is at best partially observable which implies all factors affecting performance are not directly observable. In a dynamic environment the expected performance of a provider as referred by another agent may have changed based on current load and is not necessarily an indication of the trustworthiness of the referring agent. Besides, an agent is unable to accurately assess the impact of its own decisions, both choosing service providers and making referrals, on its environment.

As such it might not be feasible to seek or determine strategies for optimizing performance. Rather, we posit that agents are concerned about finding a quality of service which exceeds an acceptable performance threshold, γ. This formulation is consistent with Simon and others view of bounded rationality of decision makers within the context of complex organizations [2,7]. Other approaches from game theory also use the notion of *aspiration levels* to stabilize systems and reach equilibrium [1,8,9,10,12].

In this paper, we provide an approach for trading referrals using which agents can locate high-quality service providers. Our proposal involves learning to rate referrers and use such ratings to adjust future referrals to identify effective service providers. We experimentally demonstrate the convergence to satisfactory service provider selections for the entire group of peer agents.

2 Framework

We now present a framework for the environment and the algorithm representing the decision procedure used by the agents to seek and use referrals about service providers.

2.1 Environment

Each agent is assigned a total load of L for each of several different task types per day. At the outset, each agent knows the set of providers that can process each task type without the knowledge of their intrinsic capabilities, or their performance functions. The agents are also unaware of the current service load of any provider.

We designate $\mathcal{E} =< \mathcal{A}, \mathcal{T}, \mathcal{P} >$ as the environment, where

- $\mathcal{A} = \{a_k\}_{k=1..K}$ is the set of agents,
- $\mathcal{T} = \{t_n\}_{n=1..N}$ is the set of task types,
- $\mathcal{P} = \bigcup_{n=1}^{N} \mathcal{P}_n$ where $\mathcal{P}_n = \{p_n^m\}_{m=1..M_n}$. M_n is the number of service providers for task type n. p_n^m is a provider which can perform a task of type t_n.

A provider, p_n^m, is characterized by a *performance* function, f_n^m, which models its performance, e.g., the task processing quality. This function should satisfy the following conditions:

- $f_n^m(0) = 0$
- $f_n^m(+\infty) = +\infty$[1]
- $f_n^m \nearrow$[2]

We assume that the performance of a provider on a particular day depends on the total workload on that day: agents which use a provider the same day will obtain identical performances. Any two agents, however, may have different satisfaction levels for the same quality of performance. We represent the satisfaction of an agent by a *subjective* function S_{k,t_n} which models the satisfaction obtained by interacting with any provider of a particular task type. To be more precise, if *perf* is the performance of provider after having performed a task of type t_n then the satisfaction of the agent a_k is $S_{k,t_n}(perf)$.

2.2 Interaction with Other Agents

When an agent needs to process a task and is not cognizant of a satisfactory service provider for the corresponding task type, it approaches other agents for referrals. The model of interaction between agents is inspired by Piaget's theory of exchange value [3]. The asking agent a_a gives its load L and the type of the task t it has to process, which are the *real value*s exchanged. The helping agent a_h answers by providing a name of a provider p and its estimation e of its quality (*real value*). Depending on the received referral with quality estimates, and its evaluation of the reliability or trustworthiness of the helping agent, the asking agent may or may not use the referral. If it decides to use that referral, it asks the referred provider to perform its task t with the load L. At the end of that day, the provider responds with a value, *perf*, representing the quality of service. *perf* depends on the provider's load for the day and on its performance function, f. More precisely, if L_t is the sum of all loads on to this provider then $perf = f(L_t)$. The asking agent computes the satisfaction s it gets given *perf* by using its satisfaction function S_{k,t_n} ($s = S_{k,t_n}(perf)$). Finally, this agent communicates this *virtual value* to the helping agent. a_a regards s as a debit to a_h and a_h regards s as credit from a_a. An agent expects others to help it in return when it has previously done. Consequently, it may have the incentive to ask for help those ones it has helped, i.e., those for which it has a lot of credits. In this model, we assume agents are truthful in reporting their satisfaction obtained from using a referred provider.

In addition to finding quality service providers via referrals, agents also benefit from learning about the quality of referrals provided by peer agents. Agents keep track of their previous interactions with others in order to identify agents who have helped, i.e., provided high-quality referrals. Correspondingly, an agent may be inclined to help those who have helped in the past and less inclined to provide referrals to others. Besides, the information recorded about referrers can be useful to estimate the quality of a provider.

[1] $f_n^m(+\infty) = +\infty$ denotes $\lim_{x \to +\infty} f_n^m(x) = +\infty$.

[2] $\nearrow$ ($\searrow$) means a function is increasing (decreasing).

2.3 Information Recorded by an Agent

An agent a_k will keep track of the following information from its interaction with others where S_i is the satisfaction obtained on day d_i:

- $\Gamma_n^m = [\,(S_i, d_i)\,]$ list of satisfactions obtained by the agent a_k from providers p_n^m,
- $C_{kk'} = \langle[\,(S_i, d_i)\,]\rangle$ is a vector of lists of its credit with agent $a_{k'}$.
- $D_{kk'} = \langle[\,(S_i, d_i)\,]\rangle$ is a vector of lists of its debit with agent $a_{k'}$.
- $\Delta_{kk'} = \langle[\,(S_i, d_i)\,]\rangle$ is a vector of lists of the difference between the actual satisfaction obtained from provider referred by $a_{k'}$ and the estimation agent $a_{k'}$ gives for that provider. This information will be useful for a_k to evaluate future referrals.

The $C_{kk'}$, $D_{kk'}$, and $\Delta_{kk'}$ vectors are indexed by task types, i.e., the jth element of such a vector contains data about interactions involving tasks of type j.

2.4 Satisfaction Criteria

As we have argued previously, in a distributed, open, dynamic environment, it might be more meaningful to seek satisficing rather than optimal performance. This is particularly true in our framework as the number of service providers, their load and hence quality, the peer referring agents as well as their estimations of different service providers can all vary over time. We present a satisficing approach to service provider selection where agents will not consider switching from a service provider if the latter's performance is above a pre-specified threshold or aspiration level. We say that an agent a_k is satisfied by the performance of a provider with performance higher than a threshold $\gamma_{k,n}$ for tasks of type t_n.

2.5 Evaluating a Referrer or a Provider

We present a recency-biased weighting scheme on past interactions to estimate the performance of a provider or the referral ability of another agent.

Definition 1 ***(Weighted mean and weighted standard deviation)*** *Let $\mathcal{S} = [\,(S_i, d_i)\,]_{i\in I}$ where S_i is the satisfaction got at date d_i. Let d be the current day. Let ω be a function such that ω is a decreasing function, $\omega(0) = 1$. We call* **mean** *and* **standard deviation weighted** *by ω the values*

$$\begin{cases} \mu_\omega(\mathcal{S}) = \dfrac{\sum_{i=1}^{|I|} \omega(d - d_i) \cdot S_i}{\sum_{i=1}^{|I|} \omega(d - d_i)} \\ \sigma_\omega(\mathcal{S}) = \Big(\sum_{i=1}^{|I|} \big(\omega(d - d_i) \cdot S_i - \mu_\omega(\mathcal{S})\big)^2\Big)^{\frac{1}{2}} \end{cases}$$

The function ω determines the emphasis placed by an agent on interactions over time. If $\omega(d) = 1\ \forall d$ then we get the classical statistical measures of unweighted mean and standard deviation. Besides, the quicker ω decreases the closer μ_ω is to recent obtained satisfaction.

Definition 2 (Weighted experience) *Let e be a function such that $e(0) = 1$, $e(+\infty) = 0$ and $e \searrow$. We call* **experience for $\mathcal{S}$,** *denoted $e_{\mathcal{S}} = [\,(S_i, d_i)\,]_{i \in I}$ the value $\sum_{i=1}^{|I|} e(d - di)$.*

Definition 3 (Class of Usefulness functions) *The* **usefulness function** *of an agent a_k (UF), h_k, are functions such that $h_k : [0,\ 1] \times \mathbb{R} \mapsto [0,\ 1]$ (the first argument is the mean of performance, the second is the experience),*
$h_k(m, 0) = 0$, $h_k(m, +\infty) = m$, $h_k(m, \cdot) \nearrow$
$h_k(0, e) = 0$, $h_k(\cdot, e) \nearrow$

The function h_k combines both the rating an agent has about another agent or a service provider and the amount of experience it has with that entity. The more the experience the more reliance the agent places on the ratings. In other words, μ_ω allows an agent to assess others. e is a measure of the amount of confidence an agent may have of its evaluation. Indeed, the more the experience the more reliance the agent places on the ratings.

2.6 Finding an Agent for Seeking Referral

When agent a_k needs a referral it tries to find an agent which has in the past referred high-quality providers and is also willing to provide referrals and have been consistent in returning help-giving behavior. Each agent $a_{k'}$ is assigned a likelihood $\omega_{k'}$ which increases with the quality $q_{k'}$ of previous referral given for the task a_k's task t_n: $q_{k'} = h_k\big(\mu_\omega(D_{kk'}[t_n]), e_{D_{kk'}[t_n]}\big)$. Additionally, a_k will be inclined to help agents which had helped previously. Let $bal(a_k, a_{k'})$ be the difference between all credits and all debits of a_k from $a_{k'}$. $bal(a_k, a_{k'}) > 0$ means a_k has given more information to $a_{k'}$ than it has received from it. An agent will prefer agents with which it has more credit: $\omega_{k'}$ increases with $bal(a_k, a_{k'})$.

2.7 Performing a Task

When agent a_k needs to perform a task, it evaluates the expected satisfaction for a provider p_n^m as $S_n^m = h_k(\mu_\omega(\Gamma_n^m), e_{\Gamma_n^m})$. It will choose the provider with the greatest expected satisfaction if that is greater than γ_k.

If a_k thinks it may not get the satisfaction it expects given the information it has collected previously, it will ask other agents for referral. A referrer $a_{k'}$ is chosen as described above (Section 2.6). $a_{k'}$ is approached by a_k as described in Section 2.2. $a_{k'}$ answers by given an estimation of the quality $q_{k'}$ of the referral it is giving. a_k will correct $q_{k'}$ by using the information it has collected in $\Delta_{kk'}[t_n]$ (defined in Section 2.3). Let ϵ be a number chosen from Gaussian distribution

$\mathcal{N}\big(\mu_\omega(\Delta_{kk'}[t_n]), \sigma_\omega(\Delta_{kk'}[t_n])\big)$. $q_{k'}$ is corrected by adding ϵ to it. a_k corrects the estimation of $a_{k'}$ since the way of evaluating a provider is not necessarily the same between two agents (different satisfaction functions). Besides, it helps also to correct possible deception attempts. The referral is chosen with probability $q_{k'}$. If it is not chosen, another agent will be approached. If no provider has been chosen when all agents have given their referrals, then a referral will be chosen among those previously referred with likelihood $q_{k'}$.

Once a provider is elected, it is asked to perform the task with load L. At the end of the day this provider will return a value of the performance $perf$ with which it performed the task. $perf$ is computed with the performance function f with the sum of all loads ordered during the day as parameter.

2.8 Responding to a Request for Referral

When approached for referral, we assume that the helping agent refers the best service provider known to it for the corresponding task type. Therefore, the agent refers the provider it thinks is the best given its estimation for the expected quality of service. We also limit the effects of chains of referrals on the stability of the system by imposing the restriction that an agent is not allowed to refer a provider it previously received as a referral.

3 Characterizing Satisficing System States

In this section, our objective is to characterize preferable distribution of agents over providers based on agent and provider parameters like S_{k,t_n}, f_n^m, etc.

In this initial study, we restrict ourselves to a specific variant of the general model we have described above:

- the number of providers is constant equal to M over each type
- the load of an agent is the constant, L, for each type and for each day and is equal for all agents
- the satisfaction function S_{k,t_n} and the aspiration level, $\gamma_{k,n}$, is the same for each agent and for each task type and is denoted by γ.

The satisfaction obtained by an agent when using a service provider on a given day depends only on the performance function of that provider and the total load on that provider for that day.

Definition 4 (Distribution of agents over providers) *We call* **distribution of agents over providers** *for a type t_n the set $\mathcal{D}_n = \{\mathcal{A}_m\}_{m=1..M}$ such that:*

- $\mathcal{A}_m \subseteq \mathcal{A}$ *($\mathcal{A}_m$ may be empty)*
- $\bigcup_{m=1}^{M} \mathcal{A}_m = \mathcal{A}$
- $m_1 \neq m_2 \Longrightarrow \mathcal{A}_{m_1} \bigcap \mathcal{A}_{m_2} = \emptyset$

Definition 5 (Set of satisfaction of a distribution) *The set of satisfaction of a distribution of agents over providers (denoted by $\mathcal{S}(\mathcal{D}_n)$) is the set of satisfactions obtained by the agents in that distribution. More formally, $\mathcal{S}(\mathcal{D}_n) \stackrel{def}{=} \{S(|\mathcal{A}_m| \cdot L)\}_{m=1..M}$. Note that this is a set with no duplicate values.*

Definition 6 (γ-acceptable distribution) *A distribution $\mathcal{D}_n$ is called γ-bacceptable if and only if*

$$\forall s \in \mathcal{S}(\mathcal{D}_n) \quad \gamma \leq s$$

A γ-acceptable distribution is a distribution where the satisfaction of each agent from the provider it is currently using is more than its aspiration level. If agents arrive at such a distribution, they will not be willing to change their choice of service providers, and hence the system will be in equilibrium.

4 Experimental Results

We have implemented this framework by choosing the following parameters: 5 different task types, 10 agents, $L = 1$, the performance functions of providers are equal to $(\alpha \cdot x + \beta)^3 - \beta^3$ (we have used $\beta = 1$), and the satisfaction function of agents are equal to $\dfrac{1}{1 + 0.7 \cdot x^2}$.

We have run several sets of experiments by varying γ, the number of providers and the parameter α. We observe if the daily distribution has stabilized after some days and if the final distribution is γ-acceptable. The table 1 shows the set of satisfaction given the following configuration of service providers:

- 3 providers with $\alpha = 0.1$
- 2 providers with $\alpha = 0.12$
- 3 providers with $\alpha = 0.15$
- 4 providers with $\alpha = 1$

We observe that for $\gamma \in \{0.6, 0.7\}$, there is convergence to a γ-acceptable distribution. In the case $\gamma = 0.6$ there exists two γ-acceptable distributions: both of them occur in our runs and there is no particular way to predict which one will be selected. When $\gamma = 0.7$ only one γ-acceptable distribution exist and hence it is the distribution which is chosen by agents. For $\gamma = 0.75$ no stable behavior is observed. In this case, there is no γ-acceptable distribution. So, the system cannot stabilize since when some agents are unsatisfied, they prevent others from being satisfied by continuing to move from one provider to another.

4.1 Influence of γ on the Speed of Convergence

We further evaluate the influence of γ on the speed of convergence to satisfactory distributions. For each value of γ we run 10 experiments and calculate the average number of days necessary to reach the convergence (see Figure 1). We can see that for small values of γ the convergence is reached very quickly. In these cases,

Table 1. $\mathcal{S}(\mathcal{D}_n)$

γ	$\mathcal{S}(\mathcal{D}_n)$ for Type i
0.6	– $i = 1$ {0.93, 0.9, 0.84, 0.73} – $i = 2$ {0.93, 0.84, 0.73, 0.63} – $i = 3$ {0.93, 0.9, 0.84, 0.73} – $i = 4$ {0.93, 0.9, 0.84, 0.73} – $i = 5$ {0.9, 0.84, 0.73, 0.63}
0.7	– $i = 1$ {0.93, 0.9, 0.84, 0.73} – $i = 2$ {0.93, 0.9, 0.84, 0.73} – $i = 3$ {0.93, 0.9, 0.84, 0.73} – $i = 4$ {0.93, 0.9, 0.84, 0.73} – $i = 5$ {0.93, 0.9, 0.84, 0.73}
0.75	No convergence

the number of γ-acceptable distributions is high and so is the probability of finding one which leads to quick convergence. However, when γ gets close to 0.75 the number day required to find out a distribution where every agent is satisfied increases significantly. This is explained by the fact the number of γ-acceptable distributions become very small in this range: 2 for $\gamma = 0.65$ and 1 for $\gamma = 0.7$.

4.2 A Case with a Continuous Load

We also ran experiments with the same configurations as before but with a daily load which follows a normal distribution with a mean equal to 1 and a standard deviation equal to 0.5. We wanted to compare convergence results in this case with the constant load situation. We ran a 300 day simulation and calculated the average daily load of each provider for the last fifty simulation days. We present in Table 2 the average satisfactions obtained by each agent. We can see that when $\gamma \in \{0.5,\ 0.6,\ 0.65\}$ the average satisfaction is "γ-acceptable". Consequently, we can expect some kind of convergence for the continuous case.

5 Related Work

Referral systems have recently received increasing attention among multiagent researchers. In [11], Yu and Singh study a referral system when an agent helps the human user find relevant expertise and protect him/her from too many irrelevant requests. In our previous work we have studied the use of referrals to locate service providers when an agent first enters a new community with

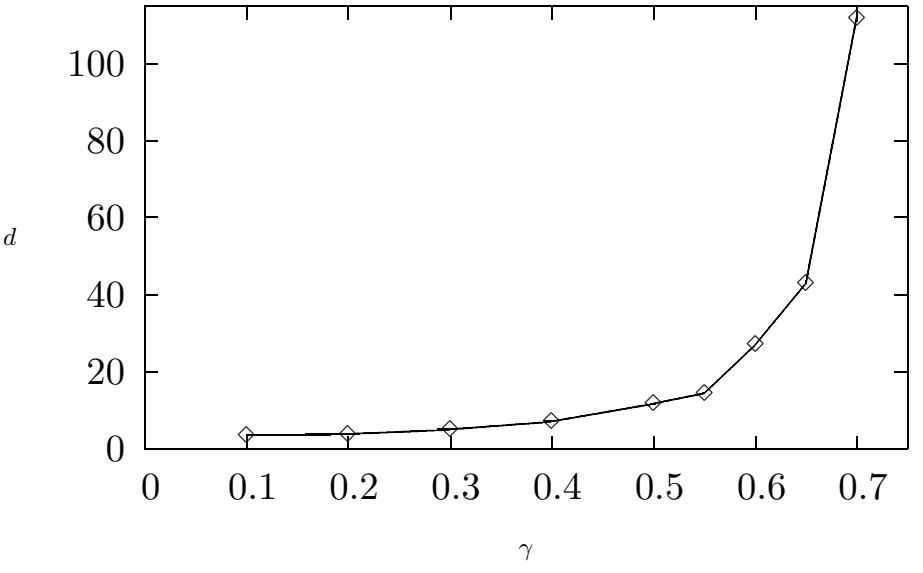


Fig. 1. Effect of γ on the number of days required to reach convergence

Table 2. Satisfaction obtained by agents with a continuous daily load

γ	Average satisfaction
0.5	{0.88 0.77 0.72 1 1 0.90 0.89 0.83 0.83 0.81}
0.6	{0.84 0.80 0.79 0.91 0.97 0.82 0.82 0.83 0.87 0.88}
0.65	{0.85 0.85 0.77 0.81 0.76 0.99 0.99 0.75 0.79 0.87}
0.7	{0.87 0.83 0.83 0.28 0.65 0.83 0.88 0.90 0.83 0.88}
0.8	{0.880.87 0.88 0.065 0.072 0.87 0.89 0.87 0.86 0.82}

no prior knowledge of the quality of service providers or the reliability of the referrers [5,6]. The use of exchange values, motivated by Piaget's theory, for social reasoning in artificial societies is discussed in [4].

6 Conclusion and Future Work

We have presented a framework for agents to choose high-quality service providers by utilizing referrals from other peer agents. We assume that the quality of service provided depends on the intrinsic properties of a provider and the total workload it faces on a given day. Our model accommodates different satisfaction levels for different agents for the same provider performance. We are primarily interested in agents making satisficing choices in that they do not change providers as long as their satisfaction exceeds an *a priori* aspiration level.

Agents need to learn both the quality of service providers as well as the capability of other agents as referrers. Experimental results confirm convergence

of agent populations to service providers in satisficing distributions for a large range of aspiration levels, both for constant and varying task loads for agents.

We are currently working on theoretically characterizing requirements for convergence to satisfactory distributions. This paper makes some simplistic assumptions of agent truthfulness while reporting satisfaction to referrers, providing best referrals when asked, always responding to request for referrals, etc. We plan to investigate more complex scenarios involving exploitative and deliberately malicious agents in the population.

Acknowledgments. This work has been supported in part by an NSF award IIS-0209208.

References

1. G. Kersten. Nego-group decision support system. *Information and Management*, 8:237–246, 1985.
2. J. G. March and H. A. Simon. *Organizations*. John Wiley & Sons, 1958.
3. J. Piaget. *Sociological Studies*. Routlege, London, 1995.
4. M. R. Rodrigues, A. C. da Rocha Costa, and R. H. Bordini. A system of exchange values to support social interactions in artificial societies. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 81–88. ACM Press, 2003.
5. S. Sen and N. Sajja. Robustness of reputation-based trust: Boolean case. In *Proceedings of the First Intenational Joint Conference on Autonomous Agents and Multiagent Systems*, pages 288–293, New York, NY, 2002. ACM Pres.
6. S. Sen and N. Sajja. Selecting service providers based on reputation. In *Proceedings of the AAAI-2002 Workshop on Multi-Agent Modeling and Simulation of Economic Systems*, 2002.
7. H. A. Simon. *Models of Man*. John Wiley & Sons, 1957.
8. J. L. Stimpson, M. A. Goodrich, and L. C. Walters. Satisficing and learning cooperation in the prisoner's dilemma. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 535–540, 2001.
9. R. Tietz and O. Bartos. Balancing of aspiration levels as fairness principle in negotiations. In R. Tietz, editor, *Aspiration Levels in Bargaining and Economic Decision Making*. Springer-Verlag, Berlin, 1983.
10. H.-J. Weber. Theory of adaptation of aspiration levels in a bilateral decision setting. *Zeitschrift für die gesamte Staatswissenschaft*, pages 582–91, 1976.
11. B. Yu and M. P. Singh. Searching social networks. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 65–72. ACM Press, 2003.
12. S. Zionts, T. Stewart, and V. Lotfi. An aspiration-level interactive model for multiple criteria decision making. *Computers in Operations Research*, 19(7):671–681, 1992.

Towards an Emotional Decision-Making

Mickaël Camus[1,2] and Alain Cardon[2]

[1] L.E.R.I.A.
{Epitech.} 24 rue Pasteur
94270 Le Kremlin Bicêtre, France
[2] LIP6 UMR 7606 Paris VI,
UPMC 4 Place Jussieu, 75252 Paris Cedex
{Mickael.Camus,Alain.Cardon}@lip6.fr

Abstract. Industrial need new technologies to make evolved methods, strategy and capacity. Decision-making plays an important role in the case of the robot evolving in an instable environment, and enables us to decreasing the human factor in the control system. This paper presents a model based on brain construction to give a restricted emotion to a machine in order to control a multitude of entities. Emotion permits to make a rapid decision in a hostile environment. The method used to build this system is based on Massive MultiAgent System (Massive MAS). It enables us to have a vast number of entities with an asynchronous communication.The morphologic aspect is used to observe the agents behavior with the aim to generate a restricted emotion in order to make an action plan.

1 Introduction

New technologies help industrial to develop knowledge in communication, exchange, security, cryptography and others. Data and decisions are critical, that is the reason why all computing tasks are managed by a human factor.

A good example is the Unmanned Aerial Vehicles (UAV) to search information. This technology is controlled by humans, but the dangerous task is managed by the machine. It is an interesting technology but it is hard to be used in an hostile environment. It is a reason to give a more important role to a computer. This decision allows to mimic a human in a decision-making process. There is important needs in computer science to succeed in this ambitious project because dataflow is very important in an unstable environment in constant evolution. A good solution would be a control system software based on human emotions [1] for the decision making, with a simulation section for the training and the composition of autonomous. Some work proved that emotions have a link with human decision-making such as Bechara's work presented in [2] or also Lerner's work presented in [3]. A software with these features is in development, its codename isPALOMA. It is developed with the oz/mozart system [4] [5] to have a maximal concurrency and a high message passing. This paper focuses on the decision-making model.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 23–32, 2006.

2 Aims

The human brain process different pulse and allow human-use of nervous system. Pulse have a role in the knowledge processing which is saved in the gray matter, and it is proved that emotion is a pulse and has also a role in the decison-making. Matter, pulse and physiology are the base of the brain operation. This unit is adaptative, allows to generate decisons, emotions, create thoughts and ideas.

The goal of this project section is to build an adaptative system mirroring the emotional and inentional human behavior as suggested and developed by Cardon [1] to make decisions in an unstable environment with a multiagent system.

3 Project

The project PALOMA is developed to simulate and build an autonomous system. It provides for the construction of an environment for a particular problematic with coherence, naturalness and reality. This tool is described by Camus and El Kadhi in [6]. As a part of the global PALOMA project, this paper details the development of the decision-making level (DML) based on the brain construction for a Sony Aibo. In this DML, We rely on a particular approach based on the mirroring of the human nervous system as described by the neuroscience research community [7]. The goal of this section is to build a system allowing an intentional behavior based on emotions.

4 Machine Representation

In this work, a machine is represented as an entity with two levels: hardware and software. These two levels cooperate and evolve together. The hardware is composed of multiple sensors and effectors. The software is, in fact, the brain of the hardware. It is not embedded in the machine, but is installed on a workstation where several calculus of the software can be distributed on a network, such as in the figure 1. In an Aibo, we have engines for legs, a microphone, a video camera associated with different sensory sensors and distance sensor to interpret a scene. Each sensor and effector has a role and measures a specific action in an environment. The “brain” component for the Aibo application is, for us, a multiagents system gathered in a polymorphic geometrical form that represents the emotional behavior of the entity during a decision-making event.

5 Multiagent System Architecture and Aspectual Agent

MultiAgents System enables us to mimic the complex mechanism of the generation of emotions [8]. To reach this goal, different roles are defined. They allow the data process with importance and quality priority. The term used here is “agentification" in french, the effectivness system depends on this action.

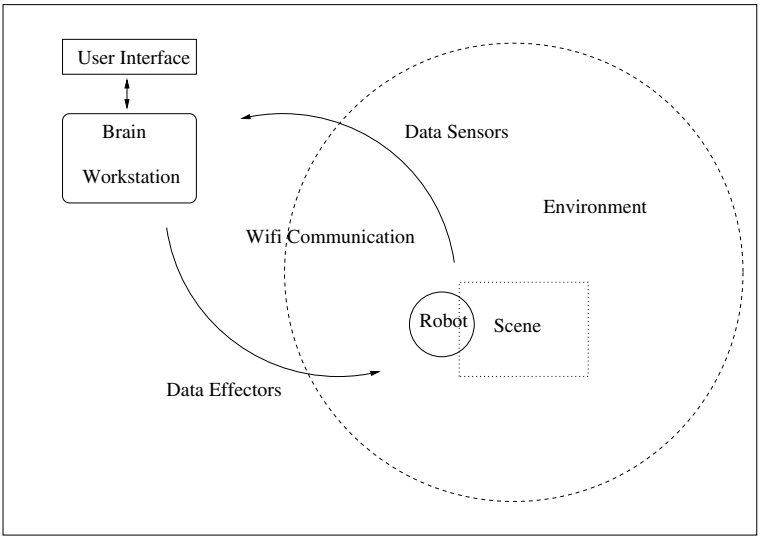


Fig. 1. Workstation and environment

Definition: *A role is related to a knowledge base. It is for example, the recognition of a specific object saved in the knowledge base in the environment.*

Agent structure is common to all the MAS, only their role differs. A role is a domain of knowledge, there is different group of roles in the system. This structure is presented in figure 2. It is simple because the important points are the system organization, the roles, the agents number and not the agent complexity. An agent is composed of a knowledge base with intentions and goals, with a communication API and an inference engine.

Definition: *A goal is a search action to find values which are in the domain of the knowledge base.*

Definition: *An intention is a list of actions to succeed a goal.*

Knowledge base is crucial, it directs an agent towards a specific role. For example, in a UAV, an agent can be interested in the obstacle detection and not in the threats on the ground. These threats will have a particular interest for other agents.

A central element in this architecture is the agent state. The different states are represented by a dynamic graph, with a history, and the possible states. The actual state is transmited towards agents with different roles.This state is conform to the knowledge base. This information transit is very important, it is the core of the system. The communication API must be powered to decrease the latency time between requests and answers.

The agent knowledge allows to have a state on process actions. The processing of actions are called "intentions". After a data processing, an agent has intentions to reach

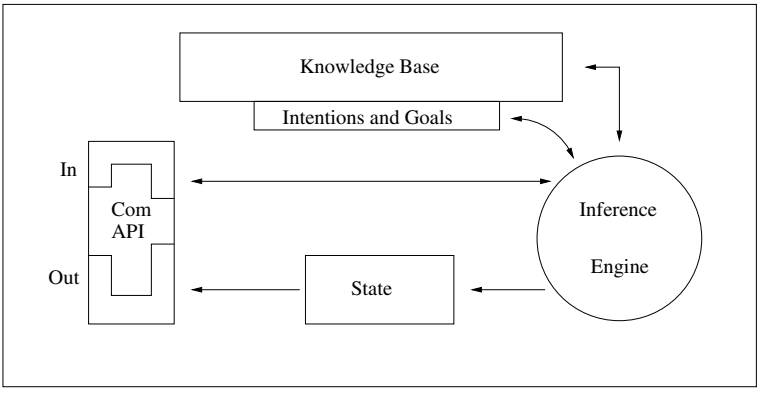


Fig. 2. Architecture of the aspectual agent

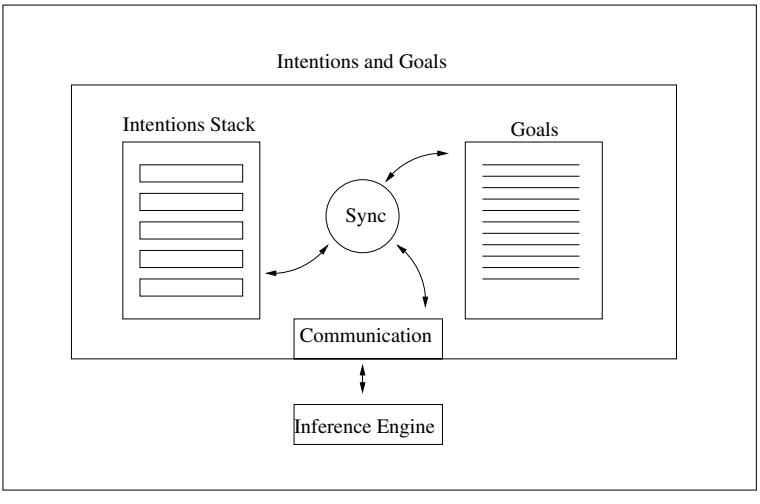


Fig. 3. Goals ans intentions

a specific goal. This system is presented in Figure 3 where we can see an intention stack and a list of goals. These two elements are synchronized and communicate directly with the inference engine to treat the base.

6 Brain Features

Let us, now, specify the capacities of the "brain" system. To evolve cleverly in an instable environment, a robot must be able to collect data, analyze, make-decisions and then act according to its goals. This is the natural human behavior such as presented in

[9] by Kolb and Whishaw. Our approach to decision-making in an hostile environment allows us to divide these capacities in five crucial levels:

1. Represent a contextual situation.
2. Direct the attention on particular elements (objects or actions).
3. Feel emotions based on these elements.
4. Build behavior action plans.
5. React to the action feedback of an object.

These levels are developed step by step with a multiagent system vision and are detailed in the following sections.

6.1 Description

The first level is a scene representation. Aibo sensors, such as sensory sensor, distance sensor and the video camera, allow to process environment data to give a contextual position to the Aibo. The goal of the camera is to recognize some simple forms such a door, a wall, a chair and others. The camera data are processed by a neural network embedded in any agents of the system in order to build a little vision ontology. This vision ontology is associated with the sensors ontology and is distributed in the multiagent system with different links.

The second level associates the goals of the robot with its environmental knowledge. The goal is to give priority to some objects or actions in the environment. Let us take an example: if the goal of the Aibo is to be near its owner all the time, the processing priority is the movement of the owner for the action, and, for example, the shoes of the master for the object. This method allows us to select data in the information flux to increase the system treatment.

The third level works on the multiagent system morphology to detect particular geometrical forms stable in a timeline. This form is analyzed by a method presented by Campagne in [10]. It allows us to recognize and classify the different geometrical forms as an emotion generated by the robot during its evolution in the environment and during the recognition of objects and actions.

The fourth level creates a relationship between the cognition and the action. The word "behavior" is used for the "reactions" of the robot to the environment (the robot uses its effectors to act on the environment). For a cognition degree, there is a succession of actions on different effectors. More the cognition degree is high, more the list of actions is specified.

The fifth level is a continual and infinite bidirectional interaction and adaptation between the environment and the robot behavior. For each action, there exists feedback, a relationship between sensors and effectors. The attention of the robot evolves with the actions processed in the environment. Cognition and action are treated in parallel in the multiagent system.

6.2 Organisation

The organisation of the multiagent system is based on the "aspectual" agent type presented by Cardon in [11]. It is a multi-level representation. In the case of an Unmanned

Aerial Vehicle (UAV) simulation for example, UAV is not limited at an agent, but is represented by a multiagent system. It represents a situation by features (called structuring organisation which is also presented in [11] and [1]), interpretes a scene to make decisions, and finds the adapted moment to act on the environment with a continual utilisation of the morphologic control. The system evolved in the environment with a systemic loop:

sensors $\longrightarrow$ **representation** $\longrightarrow$ **effectors** $\longrightarrow$ **sensors**.

An emotion is generated with an emergence on several knowledge. Emotion is not a simple variable with an applicative field such as fuzzy logic, but rather an agent processus with a particular activity form in the structuring organization of the multiagent system. There is a strong analogy on the operating mode and the data exchange between the meso-limbic and the neocortex of the human brain.

A human or any other entity evolving in an environment treats all information simultaneously. The DML presented in this paper obeys the same constraints and suggests a parallel information treatment and communication. This theory is exposed by Kolb and Whishaw in [9] where authors present the different links between brain and behavior in general. Another major fact is the representation of links between sensors and effectors. For our case, this representation ensures the exchange between hardware and software. We suggest gathering our system components (sensors and effectors) according to a simple record based on sensor and effector capacity. This method allows us to add or delete sensors or effectors dynamically to increase or decrease the global capacity of the system.

6.3 Message Passing and Semantic

Exchange data and organisation can be processed with a communication between agents. This communication is specific for each role type. Oz language allows us to specify a particular design to manage, create, or delete messages linked to a role. With this exchange, several agents have acointance with others and we can observe the communication between agents to note an emergence on one or several roles [11]. The communication is crucial. We used semantic in thread message passing to mirror pulse, physiology and matter used in the brain. We can have questions on the semantic using such as: how a word is represented in the brain? where this word is saved? how uses this word to compose a sentence? Several research have proved that the brain is a continue and parallel serial image [12]. This image is view in tri-dimension, it is a four-dimension view of the brain. This processing proved that a word, a knowledge, is not saved in a particular region of the brain. All the knowledge is distributed in the brain and we used pulse, physiology and the matter to create a specific geometrical form adapted to a context in the environment [13] and communicated at ohter person with the language. More a person has vocabulary, more it could explained it. Here, semantic is used to build a definition, an abstraction of an emergence. This emergence is a strong link between different knowledge distributed in the system. This knowledge is represented by a tri-dimensional graph[1] with different distance between words[2]

[1] This graph is a parallel view of the general communication between agents in the system.

[2] A word can be an object such as a ball, a keybord, or a verb such as “Play”, “Work” and other. It is possible to have an adjective such as “Beautiful” or “Tall”.

7 Results

Severals levels in the developpement have been checked such the control system based on the morphology, the asynchronous communication between agents and the systemic loop.

The theory of the control system has been validated by the defense of a Philosophia Doctor thesis by Campagne. The title of his thesis is in english: "Multiagent System and Morphology" [14]. This theory was been applicated with the language Oz.

The asynchronous communication between agents has been developped with the concurency feature of the Oz/Mozart system. This feature is composed of "thread" and "message passing", tow tools directly include in the language. There is different type of message in the organisation of agents. A mutiplexer has been developped to manage all messages in the developped system. With this tool, it is easy to send a message towards a specific agent or a specific group. Figure 4 shows the multiplexer and its role in the communication system.

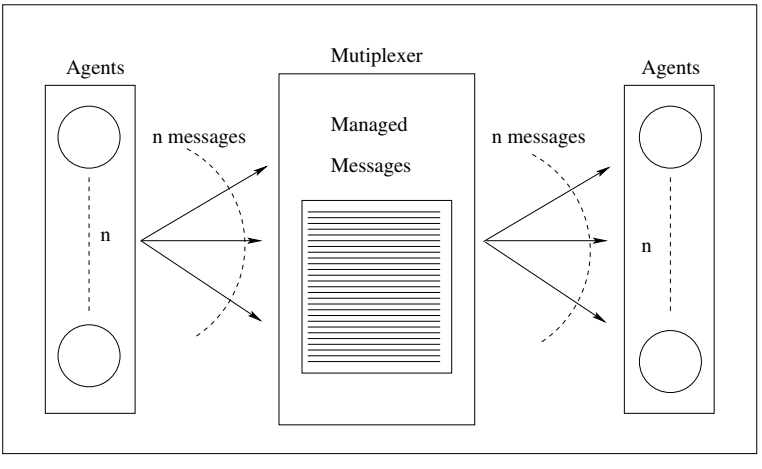


Fig. 4. Communication system between agents

The systemic loop allows us to treat data of different Aibo sensors to build a representation of the scene, generate a restricted emotion, make a decison and applied this decision with the Aibo effectors and evaluated the result with the processing of the return data. This result show us a specific organisation, a knowledge emergence such as in the figure 5. Currently, the movements of the Aibo are limited besause the ontology is tiny at this time. Now, the work is to increase the ontology and linked this ontology to more capacities of the robot.

Tools are developed for the DML, such as a user interface with an "Opengl" representation of the agents (Figure 5), to create different experiences and follow the evolution of the geometrical form generated by the structuring organization: an artificial mental

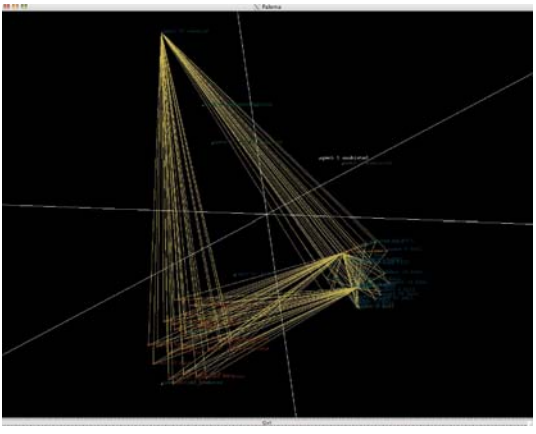


Fig. 5. Communication between agents

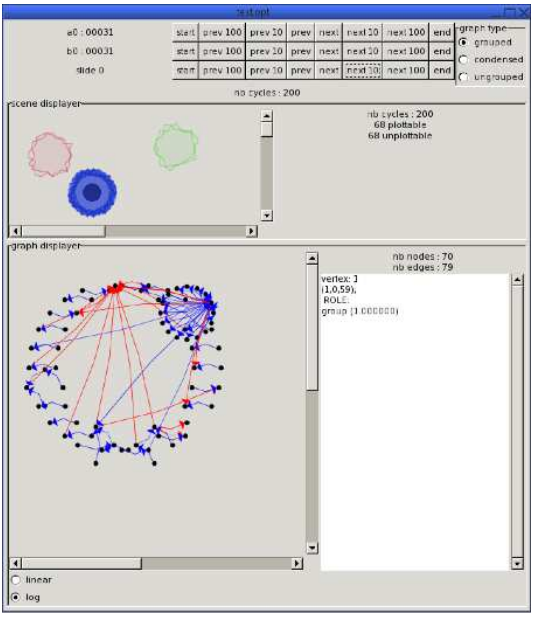


Fig. 6. Geometrical form

represention of the emotion such as shown in the figure 6. It is possible to give different parameters to the application to build the system such as the sensors, effectors and agents number. The different links between the agents, or the control method for the communication in the multiagent system.

8 Conclusion

Decision-making without human factor is crucial for an autonomous entity such as a robot, an unmanned vehicle, or a server in an international compagny. To succeed in autonomous system, it is interressant to abord different point of view, and one of these view is to create a system which mimic the human nervous system. Mirroring the human brain is also the aim of the artificial intelligence, may be that the existent technology allow us to create this system, it is a one of the major goal of this project.

Multiagent system allows us to mimic the intentionality production in the brain. It is a specific organisation of the system with an emergence on several knowledge recognize in the environment to build a representation of a scene. This method is possible with a distribution of the knowledge base in the multiagent system. This emergence highlights a part of the knowledge of the system. If these knowledges match with the goal of the system, a decision can be made, an emotion is highlighted. This decision is not optimal, there is a continue evaluation to proved the quality of the decision and direct the system on a part of its knowledge to succeed a goal if it is necessary. All the system is processed such as a systemic loop evolving in an instable environment. Knowledge are updated in the time line, this event increase the capacities of the system.

The project is developed with the system Oz/mozart [5]. Oz is a multi-paradigm language with scripting programming, object programming, logic programming and constraints programming. It allows us to use different paradigms such as the concurrency to develop a multiagent system with asynchronous communication using the message passing, or the constraints programming to create different action plans. It will be interessant to try a direct learning to use the different sensors and effectors on a robot to transformed a hardware system to a polymorphic hardware system.

References

[1] Cardon, A.: Modéliser et concevoir une machine pensante. Vuibert (2004)
[2] A. Bechara, H.D., Damasio, A.: Emotion, decision making and the orbitofrontal cortex. Cerebral Crotex **10** (2000)
[3] Lerner, J., Keltner, D.: Beyond valence: Toward a model of emotion-specific influences on judgment and choice. Cognition an Emotion **14** (2000) 473–493
[4] Roy, P.V.: Oz/mozart environment (2002)
[5] Roy, P., Haridi, S.: Concepts, Techniques, and Models of Computer Programming. MIT Press (2004)
[6] Camus, M., El-Kadhi, N.: Generic simultor environment for realistic simulation - autonomous entity proof and emotion in decision making. Journal of Systemics, Cybernetics and Informatics **2** (2004)
[7] D. Purves G. J. Augustine D. Fitzpatrick L. C. Katz, A.S.L.J.O.M., Williams, M.S.: Neurosciences. De Boeck Universite (2003)
[8] Cardon, A.: Conscience artificielle et systèmes adaptatifs. Eyrolles (2000)
[9] Kolb, N., Whishaw, I.: Cerveau et Comportement. De Boeck Universite (2001)
[10] Campagne, J., Cardon, A.: Using morphology to analyse and steer large multi-agents systems at runtime. In: Selmas 2004 IEEE, Edinburg, Scotland. (2004)

[11] Cardon, A.: Design and behavior of a massive organization of agents. Design of Intelligent Multi-Agent Systems, Human-Centredness, Architectures, Learning and Adaptation **162** (2004) 133–190
[12] Channouf, A., Rouan, G.: Emotions et Cognitions. De Boeck Universite (2002)
[13] Thom, R.: Modèles mathématiques de la morphogenèse. Christian Bourgois (1989)
[14] Campagne, J.: Morphologie et système multi-agent. PhD thesis, Université Pierre et Marie Curie (2005) draft.

A Self-adapting System Generating Intentional Behavior and Emotions

Alain Cardon[1], Jean-Charles Campagne[1], and Mickaël Camus[1,2]

[1] LIP6 UMR 7606 Paris VI, UPMC 4 Place Jussieu, 75252 Paris Cedex 05
{Alain.Cardon,Jean-Charles.Campagne,Mickael.Camus}@lip6.fr
http://www.lip6.fr
[2] L.E.R.I.A., {Epitech.} 24 rue pasteur
94270 Le Kremlin Bicêtre, France
mickael.camus@epitech.net
http://leria.epitech.net

Abstract. We tackle the notion of self-adaptive systems in an organizational way as complex systems with strong motivated autonomous behavior leading to the emotions. The main applications of such systems are in autonomous robots. We show that we need a new approach to build such systems where we use an adaptive construction grounded on software organizations of agents, using inner loops like the feedback loops of electronic, but in the agent world. The management of the activations loops by the system itself, using the notion of shape of the loops, expresses the concept computable concept of emotion.

1 Introduction

The emotions are often weighed up like the behavior of a person dependent on his/her particular situation in environment. Because the emotions have no rational characters, expressing the aptitudes of the individuals, one doesn't take them into account in the development of the traditional data processing systems. One searched for therefore, in the data processing systems, the rational behavior, the planned, controlled and optimized characters, and one didn't interest indeed to systems having characters of real autonomy, acting rather as they want, according to their mood of the moment. But the survey of such systems is of actuality today because the data processing interests to embarked systems where groups of robots and humans must cooperate on situations that can't be previously totally planned. One interest also to distributed interactive systems with property of robustness, having some tendencies similar the emotions. The emotions are the main determinants of the conservation of life in the evolution, and we set up they are the key of the real adapted systems interacting with humans.

About the artificial emotion problem, in first, we have to specify what one understands about emotion in neurobiology. Using these scientific concepts, we then could propose an architecture permitting to build a system whose the fine behavioral characters are similar pleasure or fear of the living creatures. The architecture of such a system will be evidently very particular and won't be like a system controlled since its step of conception. We will set up the hypothesis that the generation and the use of

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 33 – 52, 2006.

emotions require a particular system, while conforming to the architecture of the living organisms, where the brain is not only a local functional organ.

We will adopt a constructivist approach, where the system generating emotions will be constructed above a system producing rational results, but with a very strong coupling the two. We make the hypothesis that an emotion cannot amount to the changing value of a miraculous variable, but will be a specific subsystem indeed, complex in an organizational way. Emotion will be seen as an organizational process and the system generating emotions should adopt a type of functioning using *a spatial representation of its functioning in time*, with multiple inner loops of process, where loops of activations will be able to own-control and synchronize themselves, to reinforce themselves or to cut down. We use, for expressing such loops, multiagent systems able to deliberately self-observe themselves geometrically in-line, property we develop using the notion of morphology of agent organizations [Cardon 1999], [Campagne 2004].

The model we propose is actually applied to an autonomous robot Aibo provided of sensors and effectors, but it also applies to all data processing systems in which inputs are continuous and where actions must be determined from a representation that the system makes itself of its situation in its environment. Using the actual technology, it is possible to built distributed systems where artificial emotional effects are expressed on the Man Machine Interface of each user, in a personalized feature.

2 The Generation of Emotions in Neurobiology

Nowadays, the generation of emotions is a well-studied problem in neurobiology. It is not merely about psychological knowledge, where we observe some behavioral effects, but we have the knowledge of the internal functioning of the brain in relation with the nervous system driving to emotional states. We will lean on the available results in neurobiology and transpose them to construct a similar system, having the same principles of architecture and functionality, but using multiagent paradigm with the notion of self-reconformantion.

In biology, an emotion is a vital function of the central nervous system that triggers to typical states. It is a psychic and physiological behavioral state produced by a neural activity from an inductive, driving to some bodily behavior. Biologists found the "center of the emotions" and can precisely describe the emotional process of the production of pleasure. The center of pleasure in the brain has the following architecture:

- The first part of the central nervous system generating emotional behaviors is the *hypothalamus*. Nervous cells of the hypothalamus control the hormonal secretions of the hypophysis and so control the hunger, the thirst, the circulating electrolyte rate…
- The second component intervening in the emotional process will be limbic system. It is composed of the tegmental ventral area, situated to the basis of the brain, and of the core accumbens, situated deeply under the frontal cortex of amygdalins lobe. Limbic system is connected to the hypothalamus by its median part. It also communicates with the neo-cortex by its lateral part. The center of the emotion is going to communicate therefore with the whole of the brain.

- The other components of the system are the epencephala and the pituitary gland that generate tendencies respectively toward a goal by the production of dopamine, and that achieve the pleasure by the production of morphine molecules.

Considering more particularly the pleasure, the called "rewards – punishments" system was localized and well clarified by neurobiologists. The steps permitting to have the generation of a pleasure emotion, are:

1. at the beginning the system is in a state of neutral normal functioning,
2. a signal, in the biologic meaning of this term, trigger the process of generation of a specific type of biochemical components,
3. in Epencephala system, at the reception of the components freed by the previous process, an incentive expressed by flux of dopamine hires the system to enter in activation toward an expressed goal,
4. a center of pleasure - displeasure, the pituitary gland, values the success or the failure of the reach of the goal stated by the incentive, generating morphine molecules, and hires the pursuit of the process or makes stop. In any case, the action is a real process that tampers limbic system by memory effects.

We propose to transpose these results in the domain of the computable data processing.

3 General Architecture of a Self-adaptive System Generating Emotions

The data processing always drives to the construction of a system running in a computer, but the behaviors of systems are varied. One can classified the systems, according to their behavior, in three categories.

3.1 The Reactive Systems and the Automatic Action

A reactive system is a system for which each event of the environment is feared like a stimulus that instantaneously puts in action in a way that strictly corresponds to the stimulus characters. There is a causal and sufficient link between outside events and the reactions of the system that are elements of the same nature. Link stimulus - action is a procedural call, a kind of reflex method. The system is constructed and programmed to answer some restricted class of events by automatic activations of procedures.

The central problem for such systems is the control and the efficiency of the action event - corresponding subroutine. This type of system doesn't attach any meaning to its actions and only an anthropomorphic observer's can attach some emotion to its behavior. These systems are, with regard to their construction, very decomposable in clearly identified parts with, for each part, a very precise functional role.

3.2 The Perceptive Systems and the Selective Action

In this case, each event coming from the environment is considered as a complex fact. There is symbolization of the event feared, representation in an internal entity

composed of many data and using knowledge basis. This entity, that symbolizes some aspects of the things of the environment, is clearly defined and its structure can be complicated. The problem is then to well recognize the whole characters of the event for the precise identification and have a reaction in an appropriate manner. Some recognition characters permit to identify the event and to drag the system reaction suitable and precise. The system distinguishes elements of the environment while symbolizing them by many predefined characters specified by ontology. It is constructed with a mediating module between environment and subsystem of action, but it remains a solver of problems. The central problem is the fusion of data, achieving the loop "event, recognized symbols, structure of such symbols, interpretation, action, event again". These systems are, as for their construction, totally decomposable [Mataric 1995].

The two types of systems, reactive and perceptive, solve the well-stated problems and they are constructed typically for that. Their possible property of training is of type backing, in view to improve their reactive performance, this one tackling the gap between real event and recognized one. But it exists a third type of system, of a very different nature. For these new systems, inspirited of the living ones, it is not anymore the main question to solve well-known problems in formal ways, but rather to experiment possibilities of self-organization of some of their components, expressing problems they formulate for themselves, and treat in their own [Brooks 1991], [Dautenhahn 1997]. Such a system, when it copies the behavior of a living organism, must have the same reason to adopt some behavior that the living organism it simulates. These systems, called the *self-adaptive systems*, have radical differences with the two previous ones [Cardon 2004].

3.3 The Self-adaptive Systems and the Motivated Action

A self-adaptive system is a system composed of two different strongly linked parts [C.f. Fig 2]:

- a substratum part, rational in way, managing the inputs and the reactive effects as well as the logical and rational automatic actions,
- a specific part deliberately representing the current situation of the system in its environment, according to some subjective characters and controlling the substratum part.

Such system is active for itself with the means of its structure indeed. The system part representing the current situation will be constituted of many entities in permanent reorganization action, adapting this internal organization at a time to the actions towards the environment and also to its own organizational state, like in the brain.

Self-adaptive System

> *A self-adaptive system is a system composed of a rational substratum and of a sub-system of representation of the current situation, controlling the substratum and formed of entities having the capacity of adaptive reorganization. This subsystem of representation adapts its organization at a time with the state of the environment, and so with its organizational state, following its own tendencies.*

Such a system can continuously construct representations of events, according to reasons that will be its own, according to its specific situation in its environment,

according to the possibilities of its structure. The notion of representation, the internal object that represents a scene or a thing of the environment, is first. The notion of adaptativity is considered in the strong sense therefore. Determinants of this representation are constituted the first, according to the capacity of the system, according to its organizational memory, under some stimuli, while putting an aim toward something it takes into consideration. The system notices something that it aims. It conceives a situation, elaborates some plans, and after acts. The construction of such a system is not merely decomposable in very precise functional parts, because its active parts can hold roles that evolve during the generation of plans. Its fundamental property is the re-organization of its active parts [Dretske 1988]. A self-adaptive system can evidently degrade its structure in the one of a perceptive system, while tampering its organization and while having an automatic behavior.

The Notion of Representation in a Self-adaptive System

> *An artificial self-adaptive system, before to act, builds a purposeful representation of its environment, according to some own points of view. It acts while using this representation systematically. It set up itself in adequacy with its environment, in the sense that its representations have the tendency to consolidate its global internal state for adequacy with the evolutionary characters of the environment.*

Let's note that the three systems we have presented form an architectural hierarchy: A self-adaptive system contains effectively subsystems that are only perceptive, every perceptive system containing some strictly reactive components. This hierarchical character corresponds to the evolution of the living organisms and also corresponds to the evolution of data processing systems towards the artificial life.

The part of the self-adaptive system that will produce representations and will use emotions will be composed of two distinct parts [C.f. Fig. 1], as the brain in the body: a substrate sub-system and a specific component that represents the situation and emotions at a time. These two parts are:

- a reactive sub-system, the substrate that will seize information coming from the environment and will execute standards actions. It will regroup the Man Machine Interface (MMI), sensors and effectors, bases of knowledge and functional modules.
- a sub-system for the global control. This second sub-system has two parts:
  - a sub-system generating the current representation of the situation, that will be charged to build an effective representation of the environment and to construct the current plan of action, using composition of local plans.
  - an emotion generator sub-system, strongly coupled to the previous one that will explicitly express the fundamental emotional tendencies, while tampering the production of the representation system, altering the plan of action of the system in a subjective way.

Let's note that the self-adaptive systems we consider are open in the environment and are very interactive. They accept continuous inputs and achieve output actions in efficient way. We will keep terminology "sensors, effectors" to specify inputs and outputs of the system, like in the robotic application.

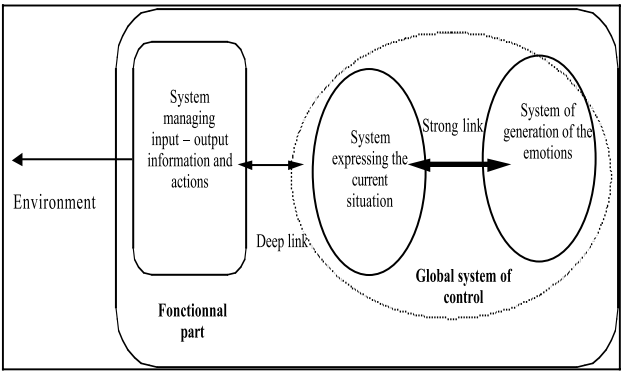


Fig. 1. Self-adaptive system and the link between representational component and the one reifying emotions

3.4 Emotional Tendencies and Adaptativity

The representational component of self-adaptive systems manages continuous structural modifications, driven by its tendencies that one must consider as unavoidable. These systems must solve adequate reaction problems indeed, but while systematically taking account the satisfaction of their tendencies.

Emotional Tendencies

> *The self-adaptive systems are conceived so that they must satisfy the imperative tendencies that drive their behavior in a decisive manner. These tendencies are in various numbers and are contradictory. They will be called the emotional tendencies. They are the global light driving the modifications of the organization of the system of representation to give a subjective tendency.*

The emotional tendencies, for the natural organisms, are emotions that advise them to survive, to feed, to reproduce, to maintain themselves in satisfactory situation in their environment... That is these reasons that drive their behaviors, first while generating some appreciable representations of their world and also allowing to act for reasons on their environment. And with the necessity to behave and to act imposed by these emotional tendencies, they are brought to solve some varied problems with efficiently. But the resolution of problems is therefore, for these systems, a mean and not a goal.

One set up that the emotional tendencies present in the self-adaptive system must be numerous and must permit opposition and choice always. In the case where the system would have only one tendency or that all of these would be strongly in agreement and constituting, for example, a hierarchy with a permanent dominant need, the system would amount to a reactive one, without tendencies. So, the system will be complex.

Self-adaptive System and Emotional Tendencies

> *A self-adaptive system is a system where the reason to function is driven by the satisfaction of emotional tendency, modifying strongly its representation of the environment. To satisfy these tendencies, the system must adapt its sub-system of representation of the situation and so modify its behavior. The structure of its representation system will be, for that reason, very plastic. The system will modify its representations while maintaining them in a domain compliant to its emotional tendencies.*

This type of adaptativity sound in an organizational way therefore, while specifying that the structure of the system of representation is fundamental to constantly drive the action. This sub-system re-organizes itself so that the global system stands in structural concordance situation with solicitations of the environment, while following pressures of its emotional tendencies.

The fundamental emotional tendencies, to be able to finely modify the current plan of action of the system, will express themselves as the characters of some organizations of software agents constituting the system of representation. Indeed, a software agent has two characters, active and cognitive; it represents knowledge and action, and is neither a particle, nor variable or process only. The fundamental emotional tendencies will be the global characters driving reorganizations of agent's organizations. It is a strong organizational hypothesis, since we will represent tendencies by types of shapes expressing the movement of agent organizations [Cardon 2004]. It is the geometric hypothesis about emotional tendency expression in large software agent organizations.

The Fundamental Geometrical Hypothesis

> *The geometrical hypothesis, with regard to the fundamental emotional tendency expression, consists to set up that the tendencies can be represented by some kinds of movements of geometric shapes in some dynamic space where speed up the organizations of software agents.*

4 The Multivalent Approach of a System with an Emotional Behavior

The architecture of a system generating emotions is radically different of an input - output one stepping levels of computation according to some predefined steps. We have to define specific inner-control of a system producing fuzzy states as the emotional ones, composed of proactive entities (entities that run for themselves), generating inner cycles of activities with specific rhythms, according to the real process of emotions in the brain and corresponding to different types of effective actions or movements. This architecture will essentially be founded on aggregation and breaking of software agents groups rather than formal neuron systems.

Expressing of an Agent Group

> *Because they are proactive, organization of software agents can represent systems in a totally organizational way, where the form of the activities and the links between agents directly lead to an effective activity of the system, in a continuously adaptive reaction.*

4.1 Basic Computable Component Producing the Emotion: The Computable Oscillator

The biologic presentation of the emotional activity describes the existence of neuron domains speeding up in loops, acting each other's for the propagation of flux of activations. We have to describe a system where the activity form is made of emergent feedback loops, positive, negative and additive. We precise the basic architectural element of the system with the following component, the computable oscillator:

Computable Oscillator

> *A computable oscillator is a software organization of agents whose activation forms quickly and by its own functioning many cycles of activity having specific intensity and speed.*

This notion spreads into the computable the one of systemic feedback loop. Such an oscillator is an organization of software agents that coordinates them, modifies the link of theirs activities, synchronizes some of them. The oscillator is formed by emergence of a self-kept structure distinct of the other agents of the organization. Such a group must emerge then to control itself, to pass from a uniform state to another where a looped process transforms a group into an oscillator. Mathematically, it is about an emergent sub-graph in the strongly coupled activation of an agent organization. Such an emergent oscillator leads to the adaptive activity the outputs of the system and must control other attempts of emergent loops.

The running system will be formed of a structured set of such elementary oscillators, permitting a global and local backing and the inhibition and the stop of the process. There is no central controller in the system but self-control distributed into the emerging components and their synchronization using negotiations. The system will function by self-control and self-regulation of its oscillators, with local limits cycles and a general faculty more or less conservative.

4.2 Typical Element of the Architecture: The Adaptive Component

One considers a system with a sub-system composed of interfaces and material components, and with a specific part generating representations and emotions, the two being very strongly linked. The generating system is composed, at the minimal level, of a set of agent organizations called the *aspectual agents*, able to easily produce by emergence groups of some computable oscillators. This architecture is typically evolutionary. From the inputs of the system, regrouped into classes according to data (the sensors for a robot), the system should produce lot of loops during its activation. There is not initial state driving automatically to a specific state of reaction, but unceasing transformations driving to progressive actions.

The very basic element of conception of the system is the aspectual agent [Cardon - Lesage 1998]. It is a software agent, typically proactive, whose role is at a time factual and symbolic. These agents serve to form some subsystems similar to those of the limbic one into the human brain. An oscillator is a group of synchronized agents that emerges and exert some power on its context. For the emergent and the control of this oscillator, we represent the behavior of every agent group by a mechanism of self-observation founded on the geometric shape of the agents' activities. We are going to associate to the notion of behavior of all agent groups the one of geometric shape. We hear shape into the classic geometric sense of the term, like hypergraphs.

Let's notice that agents being some rational entities, it is possible to associate them a precise notion of state characterized by values of specific vector.

Geometric State of an Aspectual Agent

> *The state of an aspectual agent is the meaningful characters that permit to describe its current situation at a time and to predict its future behavior. This will be a specific vector.*

It is clear that one will always bring back each of these characters to an element of R. So, an agent's state will be a point of R^m if there are m characters defining the behavioral agent's state [Cardon 1999, 2004].

Map of Activity of an Aspectual Organization of Agents [Lesage 2000]

> *A map of activity of an organization of agents is a temporal representation of the geometrical set of the significant characters of the agent behaviors. This is a dynamic geometrical object.*

To use the notion of shape, that is to represent a map of agent activity by geometric forms, it will be necessary to first represent each agent by a vector of activity. The map of activity of an organization of agents will be then a set of points in the R^m corresponding space, according to the m typical characters of each agent's behavior. The

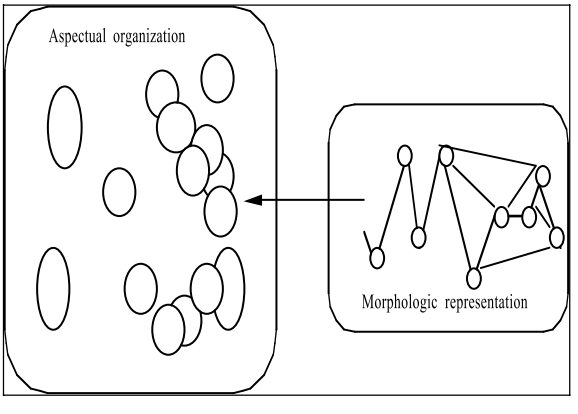


Fig. 2. The morphology expressing the aspectual agents organization

constitution of such a map is possible because agents are only rational entities. Their typical characters are expressed from their structure according to their actions only. Using a classic mathematical transformation, we express the form of the agents activities as a graph, where each node is a group of similar vectors of agents and each link is the valuation of the qualified communications between groups of agents [C.f. Fig. 2].

To represent the behavior of a self-adaptive component organization with its geometric aspects, one must define a specific dynamic space, the *morphological space* [Cardon 1999]. The so-called *agents of morphology*, distinguishing groups and extracting forms will achieve the assessment of the active shape of agent groups [C.f. Fig 3].

Finally, another organization of agents, after the aspectual and the one of morphology, is going to take in consideration the state of the aspectual organization to achieve an analysis of the aspectual agents behavior. It is about representing the sense of the activation of the aspectual agent organization with its geometrical characters and produced by the agents of morphology. The *agents of analysis* are going to provide a cognitive view of that has been expressed by the geometric and semantics information coming from the morphology agents, above the aspectual agent landscape, an interpretation of graphs indeed [C.f. Fig. 3].

Rational Functioning of the System

> *The system, reduced to the aspectual agents doing the foreseen rational tasks, with the agents of morphology expressing the shape of the aspectual activation and with the agents of analysis achieving the synthesis of the aspectual agents functioning to plan the reactions, will be qualified of rational functioning, that to be-to-say under inner-control but without any emotion indeed.*

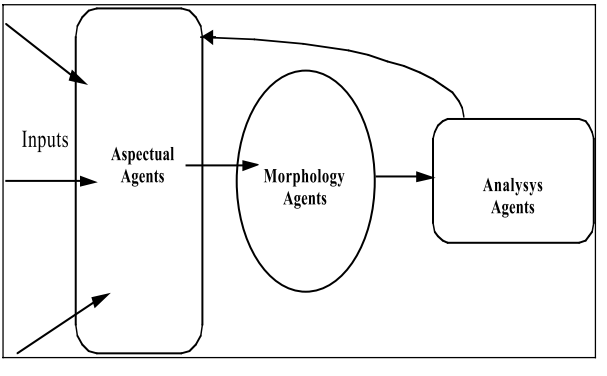


Fig. 3. The three rational organizations of a self-adaptive component

We can precise the notion of self-controlled representation with a specific coarse grain component: the adaptive component [C.f. Fig. 3]. This notion is going to permit to exceed the strictly rational behavior of the system and to define the emotions.

Self-adaptive Component

> *A self-adaptive component is an aspectual agent group linked with a morphological representation and an analysis agent group allowing to describe and to interpret its own specific activity, while controlling it in order to make emerge a computable oscillator. Such a component is emergent and does not exist at the conception level therefore.*

A linked set of self-adaptive components could express, by its functioning and in a very dynamic way, the emotional states.

4.3 Basic Organization of Self-adaptive Components: The Aspectual Agents

According to the inputs of the system (robot sensors and effectors for example), the agent organizations generating behavioral decisions and emotions will be composed of self-adaptive components, every one being an organization of aspectual agents centered on a symbolic and geometric role bound to a morphological characterization and provoking an organizational answer taking into account the information coming from its context. The behavior of this set of components will be appreciated in two ways:

- the action of agents on the output of the system (flux, intensity…), what will leads to an emotive effect.
- the shape of the organization, witch will produce an observation of emotion like a mental photography.

We base the architecture of the system on a representation of situations theory, where the reactions of the system necessarily pass by the continuous erecting of an adapted representation, the current one, that is a computable construct about the sense of the action of numerous groups of agents.

We now specify the different categories of aspectual agents. The self-adaptive components are constructed using light software agents. One kind of aspectual agents is bound to the outputs of the system (for example the embodiment parts of the robot). While preserving the robotic terminology, these agents are of two types:

- the *sensor aspectual agents*, SAA: they interpret the information coming from the environment,
- the *effectors aspectual agents*, EAA: they propose an effective action of the output parts.

Their structure is the one of a classical light agent [Krogh 1995], [Campagne 2004].

The SAA, the Sensors Aspectual Agents

They are aspectual agents that achieve an automatic interpretation "cognitive and variable in amplitude" of this one, for each new information coming from the environment in the input buffers. The knowledge module of these agents achieves the interpretation. These two modules, module of knowledge and the behavioral one, are composed of rules that are, for example, the followings:

- if value of such information increases with such contextual characters then to position the meaningful variables to such values and to get in such state…

The EAA, Aspectual Effectors

The EAA operate on the environment of the system by efficient actions (on the MMI or with the body of the robot). These agents are solicited by the aspectual and analysis ones and propose actions. For achieved these actions, the EAA must coordinate them in a social way (prey - predatory principle). Rules of action of the EAA are multilevel and are, for example, the followings:

- if the state of several solicited aspectual agents is of such category, then to make the proposition of such immediate action (with a notion of hierarchy and priority)…
- if the proposition of action is reinforced by some agents of analysis, then to act of such manner…
- if an alarm is triggered then to propose to hire such action with such priority and such speed of realization then…

These aspectual agents must define very local plans: that is the numeric synthesis of their past actions, proposed current action, possible values of future actions. Let's notice that this notion of plan is strictly local and only defined by the values of characteristic variables, permitting to memorize the details of the local actions.

4.4 The Structuring Agents and the Sets of Oscillators

The aspectual agents linked to sensors and effectors are evidently not sufficient to represent any emotion. An emotion is an internal phenomenon, in a brain or in a system. It is a particular movement of some self-adaptive components whose the general shape (the morphology) will be typical of a developed process of emotion. These self-adaptive components will be made of aspectual agents and their in-line appreciation will be a kind of game with their geometric shapes. By game, we hear that the organization of agents necessarily fears its own activity, prolongs or interrupts it.

So, we are going to consider another organization of aspectual agents, numerically and qualitatively the most important therefore, the *structuring agents*. They are particular aspectual agents without any communication with the system inputs or outputs, forming some evolutionary groups and finally operating on the EAA and SAA agents to hire a type of inner behavior that they will express in own, by their activities. Then, these inner agents are going to define very abstract functional groups, to permit the emergence of some agent's sets, active or recessive according to certain organizational characters, to put into relations some groups rather than others, to organize the activity of the aspectual SAA and EAA. They plan their functioning by the following characters:

- existence and maintains of cycles in the organization of structuring agents,
- existences and maintains of typical parts (in the geometrical sense) activated after some another one,
- existence of junction points in the graph of activities of the structuring agents,
- existence of partial order relations….

The structuring agents will form, erecting self-adaptive components, an active organization *whose functioning will be worth* for the production of an emotion. They don't have any direct relation with sensors and environment and in this way, they can only represent some "strictly abstract" conceptual shapes, like into the brain. They are going to represent events having a relation with the reality observed by the SAA, but in a geometric meaning (the sets of activated agents whose the expressed shape is typical). They have to produce some typical shapes by their activity and have to follow information of morphology agents by reinforcement and production of more typical activity shapes. They have to produce some conceptual activities, because their roles will have the characters of concepts being worth for correspondent characters of the reality. Mainly, they have to make emergence of *sets of computable oscillators*. Their activities will be continuously representative of the external phenomena given from the numeric information manipulated in the aspectual SAA and EAA. The functioning in the system is therefore the following:

- initially systematic activation of some aspectual agents SAA, EAA,
- systematic activation of structuring agents,
- morphological description and analysis of the aspectual, detection of loops for self-adaptive component emergence, creation of chains of self-adaptive components,
- backing and typical categorization of activities of the structuring agents forming self-adaptive components (choice in the system), emergence of a global typical form of activity,
- link form to form between an internal situation of agent's activity and the external phenomenon expressed (the emotional behavioral reaction) and maintain of this correspondence when activities evolve.

This correspondence form to form means that the external situation (the phenomenon) is expressed (discerned and represented) by some internal structuring organization functioning with a typical geometrical aspect [C.f. Fig 6].

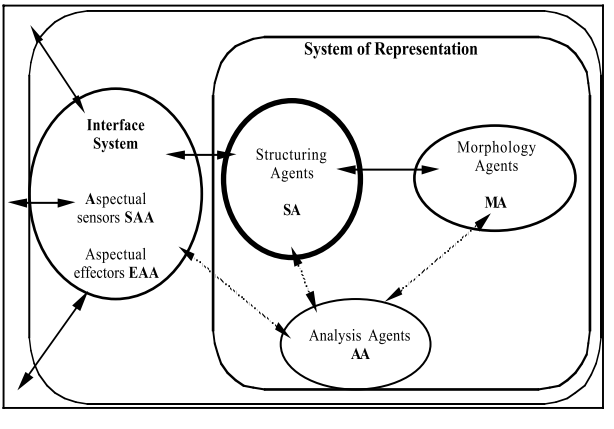


Fig. 4. The five different organizations of agents of the representation system

The structuring agents are going to concretely define, by the fact that they speed up in a coordinated way, specific activities corresponding to features of a general internal representation, a geometric sentence pushed on some characters of the current external reality. Thus, a surprise (an emotion of surprise) will see to speed up with strong vivacity some very active chains of loops of aspectual agents with spatiality role, if the phenomenon creating this effect of surprise is into the spatial domain of perception. Structuring agents have therefore specific roles to express specific categories relative to the different aspects of the phenomena in the environment.

Structuring agents represent categories that refer to:

- the space and its different possible description modes (the permanent and regular shapes)…
- the time and its modes (the notion of time that is passing out)…
- the designation of a well-identified thing (the detachment of something from a whole of shapes)…
- the situation of an object (the appreciation, the utility, the worry…)…
- the possibility to manage the organization itself (the component elements of the proto-self)…

All these general and abstract characters should be declined into different groups of structuring agents that are active according to some cases and where the characters of the groups are variable. This process will be activated according to information provided by the sensor aspectual agents SAA, or from the morphology.

5 Functioning of the System: The Motivated Pleasure

The emotional system is built upon the rational system composed in the same way of the aspectual, morphological and analysis agents, but while permitting to decide the action to undertake, and to appreciate this action in a qualitative way from a precise incentive.

5.1 Emotion

We are going to define the architecture of a specific organization of structuring agents managing the production of emotions like pleasure.

Artificial emotion

> *An artificial emotion will be essentially the rhythm of some self-adaptive components functioning in emerging way in the structuring agent organization, functioning by synchronized loops whereas the system undertakes some external typical action. In this way, the system adopts a specific in and out behavior.*

To hire an emotional process, that is a particular action of the agent organizations and typical movements of the body, it is necessary to leave from a state that is clearly a neutral one. We will call "low state" the state without emotive action or achieving an automatic one. In this state, some specific aspectual and analysis rational organizations are operational.

Low State

> *It is the state where the system is inactive or achieved a preprogrammed automatic reactive action.*

In the low state, the input of information won't trigger anything of emotional and will amount the more a reflex action. We have to give the autonomy to organizations of structuring agents, into the sense of a general functioning with inner loops. The system must adjust itself to consolidate a while its structure: some organizations of agents are in progress and will supplant some others according to a rhythm, launching some external actions according to a specific fashion. Others organizations will work then in background, without orders sent to the EAA, but they will possibly be later in an emergent state (phenomenon of domination). The adequacy between the external stream of information and the complex activation of structuring agent's organizations will allow the emergence of some self-adaptive components.

5.2 The Characters of the Incentive: From the Signal to the Incentive

We put the hypothesis that each sensor that is not a strict alarm sensor is, in some way, the corresponding of an artificial *sense* in the system. So we have:

- sensors of vision, sensors of temperature, sensor to touch or follow-up the movements…

The system fears its environment by different means that we call its "senses". To each sense will be associated, whatever it is, a state of activity:

- active or no active sense (with notion of intensity if the sense is active),
- active sense in an admissible way (toward the pleasure) or anomalous (toward the pain).

These sensory characters of the system can be easily represented into the typical structuring agents bound to some effectors agent's, the EAA indeed.

From some *signal* considered like a starting shape putting in effect the inputs of the system, there is generation of an *incentive*, that is a general tendency to the action toward some goal, when a schema of plan of action is defined into the analysis agents (global plan with local indicators of sub-plans on different structuring agents), when the system can maintain this plan some time without tampering it too much and especially while valuing it. The incentive is therefore a meaningful modifying activation of the current low state with effect in the different organizations of agents hiring to lead the system that is the bodily part and the system of representation, toward a typical behavioral activity aiming some goal. This activity can be "to seize a pen for touching it", "to drag quickly on the right of the window"… There is an engagement of an effective output activity and engagement to its organizational transposition. This signal is going to release:

- according to the state of the system,
- according to the characters discerned of the environment,
- in an irrepressible manner, but no randomly.

The signal would be generated by a specific organization of agents having no direct contact with the aspectual agents, but while taking account of the morphologies. In all the cases, the signal will be transmitted to structuring and analysis agents to hire the system in its whole toward a goal. We will represent the set up of this signal while using the fundamental organic tendency paradigm, like an emergence in a specific agent organization, taking only its information from the morphology agents [C.f. Fig. 6]. The origin of the signal is for us a virtual one.

Once this signal is activated, the system has to generate a global typical behavior. The signal is sent therefore on the organization of analysis, activating a pattern of requisite typical behavior. This pattern of behavior, in fact a plan (flight, approach, seizure, restraint, gesture of a member…) is managed by the agents of analysis (that is in context according to the current possibilities of the system) while taking information on the aspectual agents. That pattern leads to the generation of a new specific behavior taking into account the represented situation. This plan is initially very few precise, but it becomes clearer with its progression, precisely:

1. the contexts of the past of the organism and histories referring to this case,
2. the immediate undertaken action,
3. the evolved possibilities if the plan either succeeds or fails.

This global plan of action is transmitted to structuring agents and then generates a lot of specific plans (all the local parts) for the effectors agents. This generation is negotiated quickly between them and corresponds to a summary behavioral scheduling. There are therefore behavioral goals definite in a global manner, with indications driving the local injunctions (aspectual agents linked to effectors). One will say that there is setting up of an *incentive*: a goal declining into operational sub-goals in a particular aspectual agents activity. This incentive will be expressed in the specific *incentive agent organization* and will be a specific form in the morphology [C.f. Fig 5].

The Artificial Incentive

> *The artificial incentive is a global tendency expressed by a specific agent organization: the agents of incentive from the observation of some characters in the aspectual morphology. This tendency leads to a constraint on the aspectual, leads to a general planning of the action distributed into different groups of structuring agents. This planning brings about some specific plans with strong coefficient of intensity into all the aspectual agents. The incentive is represented by a form in the morphology, which is the so-called wanted form.*

Agents that generate the incentive, that causes it, are *agents of incentive*, linked to morphology and structuring agents. These agents speed up from a particular recognition sign in the current morphological organization of the aspectual agents. In return, they force some self-adaptive components to drag the system to a type of functioning while first soliciting analysis agents. They create a wanted form in the morphology the organizations have to reach.

This organization of incentive agents is always active, with more or less of intensity. It constantly observes the state of the morphology of the aspectual agents. It gives out, at the good time, a specific signal launching the process "incentive - satisfaction – pleasure".

General Algorithm

Continuous activation of incentive agents
Morphological survey of the aspectual agents
Emergence of an incentive signal into the organization of incentive agents and of a form into the aspectual morphology
Activation of the analysis organization from this signal and the form, and generation of a behavioral pattern into the analysis agents

Development of the Incentive

Generation of a typical behavior (form, goals and sub-goals)
Injunction of activation to structuring agents
Corresponding activation of the aspectual agents

It remains to define how the system is continuously maintaining the incentive in the time, notably defining a "center of the artificial pleasure". That is the realization of the emotion.

5.3 From the Incentive to the Satisfaction: The Artificial Pleasure Center

From a signal produced by the organization of the incentive agents, the system generates an incentive altering the analysis agent organization. That hires the system toward some typical behavior. We have now to define a control system for a very flexible behavior in while, permitting to adapt this behavior to maintain a general state procuring the artificial pleasure. It is not therefore to only reaching a fixed goal in optimal time, but to develop a state in the time, in an organizational way, as *an artificial pleasure*. Such a state will be, in the system, a self-controlled organizational one, that is some synchronized agent organizations functioning with loops and permitting them to maintain the system in the line of the committed action.

Commitment Toward the Satisfaction: The Pleasure

> *From a signal generated by the incentive agent organization and indicating an opportunity, the system speeds up to reach the satisfaction: it follows a type of inner behavior, with a goal fixed by itself, and modify its rhythm of functioning, if the goal seems to become closer or not leave. It maintains for a while a behavioral internal typical state: the pleasure state.*

The artificial pleasure will be the state of activity with self-kept functioning for the system, where the system is under the control of a specific agent organization, the *agents of satisfaction* [C.f. Fig. 5].

Artificial Pleasure and Agents of Satisfaction

> *A process of artificial pleasure generation is a global in while particular state where the activity of agent's organizations are under the control of the satisfaction organization. This organization leads all the aspectual agents to function with looped process with specific periods. A group of aspectual agents linked to sensors and that expresses senses is controlled by the satisfaction agents to the detriment of the others.*

An agent of satisfaction is an element that has the following characters:

- It has a classic structure.
- It takes its information therefore in the organization of incentive and analysis agents. Each agent is activated with a specific category of signal and it makes choice of an organ category it must commit to activation. So, it acts to launch the behavior of some organ having the role of sense generator, or group of organs.
- Its goal is to control the agents of analysis, and then the system in whole, while tampering their plans.
- It has a specific temporal objective that is to maintain the state of the system for a while.
- It is social in this sense that the coalition of satisfaction agents permits a coherent and fine control of all the bodily part of the system (otherwise, this control would be without coherence).

Pleasure is about therefore the characterization of the temporal activity development of a plan with assessment of two possible futures: a pleasant and the unpleasant one. Let's note that the time is here forced by the reality of the functioning of the system, the reality of the inputs, that this time is restrains by the physical behavior of the system.

When the satisfaction organization takes the control of the system, there is a strong link between the fourth agents organizations: aspectual, morphological, analysis, and the organization of satisfaction [C.f. Fig 5]. Agents of analysis and morphology are going to inform the organization of satisfaction of the progression of the imposed action, if that action is according to a typical shape of the morphology. This one is going to maintain it for a while, and then repositions itself, indicating that the quality of the satisfaction has changed. Satisfaction agents are going to systematically send messages to agents of analysis, indicating them the evolution of the satisfaction state and level that is altering the rhythm of the loops.

6 The Artificial Emotions

The principle of generation of an emotion is therefore the next one. The input sensors, via agents of interface, make to speed up structuring agents that make speed up the corresponding morphology agents. The incentive organization generates a specific form in the morphology that is to reach: the incentive morphology. The current morphology describing the aspectual activity has to transform itself into the incentive morphology. For that, the aspectual agents have to make some specific activity. The analysis agents control them in that way. The characters of the incentive and of the current morphologies are the determinants for the emotion. If the current morphology has a complicated shape, far away from the incentive morphology, the system expresses a state of tension that it is going to try to systematically reduce. This tendency to the reduction corresponds precisely to the discharge of energy that evoked S. Freud in the antagonistic action of impulses toward states of quietude [Freud 1966].

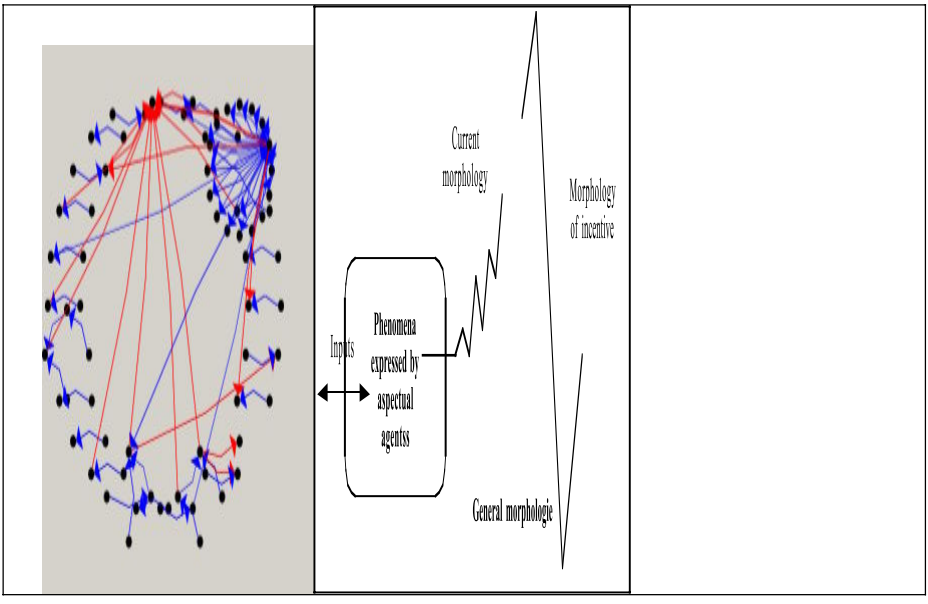


Fig. 5. The computable process of pleasure

The pleasure is the case where the current morphology reaches slowly but steadily the incentive one [C.f. Fig 7 right]. After the activity of the interface agents, the aspectual agent organization generates some global activity, a global process geometrically represented by the organization of morphology [C.f. Fig. 5, left]. The incentive morphology generates a singular pick that activated some corresponding aspectual agents. The current descriptive morphology of the aspectual agents, considered by projection in R^2, leads to the constitution of a lot of successive picks, from progressive features, and this shape converges toward the pick of the incentive morphology. In its looped functioning, agents of analysis are going to reduce the morphology to only one prominent shape. There will be reduction of tension and sensation of pleasure, as describes it S. Freud. The right figure 5 exhibits a map of the aspectual activity graph developed in OCAML language [Campagne 2004].

7 Conclusion

The emergence of emotions in an data processing system, or in a robot, has been presented as the stabilization in a while of activities in a complex multiagent system, in an organizational and constructivist way. This emergence, represented by functioning with periodic loops of activities, is the global state of a set of agent organizations. To achieve this type of functioning, some agents, the agents of morphology, represent behaviors of aspectual agents that themselves represents the minimal elements of significance. The fact that the activity of a system is endowed of emotions founds finally on a strong coupling process between the computations that organize it and the representation of computations that permits the own-control of group of agents.

The importance of such a process of coupling, binding the parts to the whole, binding groups of agents to their significance represented by agents of morphology, is strong. It is the fundamental principle of the functioning of systems that we called self-adaptive, and are to day the alone able to self-control complex systems. It generalizes the notion of feedback and systemic loop and go to the realizations of autonomous system essentially producing states by emergence.

A system generating emotions while using a bodily substratum, while proceeding to an organizational emergence thus has, according to us, a complex structure at the level of organization. But this organization can also produce a representation of itself, of its own morphology leading to the notion of "*its own body*". System can use this morphology as an engagement to act since at a time. Using geometric and cognitive aspects, that is the sign of the organizational semiotics summarizing at the same time the process of reorganization and its result. And the result of this constructive self-observation can be delivered by the system to all the observers. In this sense, such a system can express itself rather than only display values merely. The difference then, brings in such a system that expresses itself subjectively according to its intentions and another one that would proceed to displays complicated information very well adapted for sound the user, is large and makes a kind of rupture in the field, very vast, of the present computer science.

References

[Brooks 1991] Brooks R., *Intelligence without reason*, in Proc. of the 1991 International Joint Conference on Artificial Intelligence, p. 569 - 591, 1991.

[Campagne 04] Campagne J.C., Cardon A.,*Using Morphology to Analyse and Steer Large Multi-Agents Systems at Runtime*, Selmas'04 IEEE, Edinburg, Scotland, 24-25 May 2004.

[Campagne 04] Campagne J.C., Cardon A., Collomb E., Nishida T., *Massive Multi-Agent System Control,* FAABS III 2004, IEEE Workshop on Formal Approaches on Agents-based Systems, NASA Goddard Space Center, Greenbelt MA, USA, April 2004,

[Cardon - Lesage 1998] Cardon A., Lesage F., *Toward Adaptive Information Systems : considering concern and intentionality*, KAW'98, Banff, Canada, 17-23 Avril 1998.

[Cardon 1999] Cardon A., *Conscience artificielle et systèmes adaptatifs*, ed. Eyrolles, Paris, 1999.

[Cardon 2004], Cardon A., *Modéliser et concevoir une machine pensante*, ed. Vuibert, Paris, 2004.

[Dautenhahn 1997] Dautenhahn K*., Biologically inspired robotic experiments on interaction and dynamic agent-environment coupling*, in Proc. Workshop SOAVE'97, Ilmenau, p. 14 - 24, september 1997.

[Dretske 1988] Dretske F., *Explaining Behavior*, MIT Press, 1988.

[Freud 1966] Freud S., *The Complete Psychological Works of S. Freud*, J. Strachey, The Hogarth Press, London, 1966.

[Lesage & al. 1999] Lesage F., Cardon A., Tranouez P., *A multiagent based prediction of the evolution of knowledge with multiple points of view*, KAW' 99, Communication publiée dans les actes, Banff, Canada 16-22 Octobre 1999.

[Lesage 2000] Lesage F., *Interprétation adaptative du discours dans une situation multiparticipants : modélisation par agents*. Thèse de l'Université du Havre, Décembre 2000.

[Mataric 1995] Mataric M., *Issues and approaches in design of collective autonomous agents*, Robotics and Autonomous Systems, 16: 321 - 331, 1995.

[Thom 1972] Thom R., *Stabilité structurelle et morphogenèse*, W. A. Benjamin, INC, Reading, Massachusetts, USA, 1972.

A New Parameter for Maintaining Consistency in an Agent's Knowledge Base Using Truth Maintenance Systems

Qutaibah Althebyan and Henry Hexmoor

Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, Arkansas 72701
{Qaltheb,hexmoor}@uark.edu

Abstract. In this paper, we present a new and novel way of maintaining the consistency in an agent's knowledge base relying on the notion of Truth Maintenance Systems. In our work, we present a new parameter that helps in determining which propositions or assertions to retract if a contradiction ensues. Our new soundness parameter will be very easy to compute, at the same time, it is very effective.

1 Introduction

When considering the knowledge base properties of an agent, we may have in mind completeness, efficiency, consistency, among many other properties [1]. Herein, we will limit our attention to consistency. In this paper, we will concentrate on maintaining the consistency within an agent knowledge base. Particularly, we will address local knowledge base consistency in an agent, overlooking the global consistency temporarily. So, when we posit that an agent is consistent, we mean that it is consistent, regardless of the system as a whole. Moreover, if we have a group of agents, all the agents may be consistent locally. However, this does not provide the global consistency. In a system of agents, we can anticipate widely diversified information. For an agent A, a proposition S will be valid according to her knowledge, but for another agent B in the same group or system, proposition S may not be valid. On the contrary, the negation of S (not S) may be valid for the agent B, leading to an inconsistent system by having a contradiction (i.e., a proposition and its negation). In our work, we are using propositions and assertions [2] to equally mean the same. Hence, both concepts are used interchangeably in this paper. However, we concentrate our current work at the individual agent level, where a proposition and its negation coexist in the same agent, causing the agent to be inconsistent.

Section 2 emphasizes the motivation of this paper by elaborating on our big problem. Section 3 introduces some background material about Truth Maintenance Systems, and draws on some examples on how TMSs work. Section 4 gives a specification of our model. Section 5 introduces and illustrates our new novel soundness parameter. Section 6 frames the components of our model. Section 7 displays a scenario of a conflict that may arise in the system, and how the system can resolve this conflict. Section 8 draws conclusions and shows some future research directions.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 53 – 64, 2006.

2 Motivation

In our current research, we are concentrating on finding ways to enhance trust among a group of agents, and assure the information being spread among the interacting agents. We, combining the notion of trust with the notion of Truth Maintenance System, obtain new and promising results. While studying the TMS concepts, we found that the proposed way for propositions' retraction presented by Doyle and others [8] does not meet our requirements. It will be hard to involve the Retraction process in our work. So, we propose a new and a novel parameter that is very easy to calculate, give good results, and fits our requirements. In this process, we divided our work into several steps. The first step is to maintain the consistency within each agent. The details of this work will be discussed in this paper.

In the future steps, we will uphold the consistency among the agents by enforcing the global consistency. We will also combine Trust with Truth Maintenance Systems to assure the information among the agents and to enhance the trust level of agents using our novel parameter.

In a group of interacting agents, we are working in making their interaction very trusted. So, we enforce the global consistency among the group of agents. Such consistency makes assuring the information much easier. In working towards achieving the global consistency, we found that it is much easier and more effective if we maintain each agent consistent at all times. This will take care of the local consistency in each agent. We achieve this by eliminating any contradiction that may arise by retracting some of the assertions/propositions that cause this contradiction. For this end, we find a new and easier way for overcoming any arising contradiction by presenting a new and novel parameter which guards the process of retraction. In what follows, we discuss this in more details.

The following example will motivate and illustrate our work:

Consider the following group of interacting agents:

Figure 1 shows a group of interacting agents. Each agent has a local TMS that takes care of any contradiction that may arise. Also, a global TMS exists which takes care of any contradiction that may arise among the interacting agents. The Global TMS guarantees global consistency. To explicate, it is almost impossible for interacting agents not to have some contradiction, because each agent has her own vision of information. The difference in opinions always leads to some kind of contradictions.
In addition to taking care of global consistency, the Global TMS takes care of assigning reputation values for agents. If a contradiction happens in an agent, it will be detected by her local TMS which in turn, will notify the Global TMS. The Global TMS will track the chain of transactions till she gets to the source of the contradiction. It will retract some of the propositions, and send an update message to the local TMS, ordering her to both update her Database with the new action, and the trust value. The agent's local TMS will decrease the agent's trust value with the agent that caused the contradiction.

If agent A has a trust value with agent B, and the trust value becomes less than the threshold value, then agent A will consider agent B as a not-trusted agent, and she will not accept any new propositions from that agent.

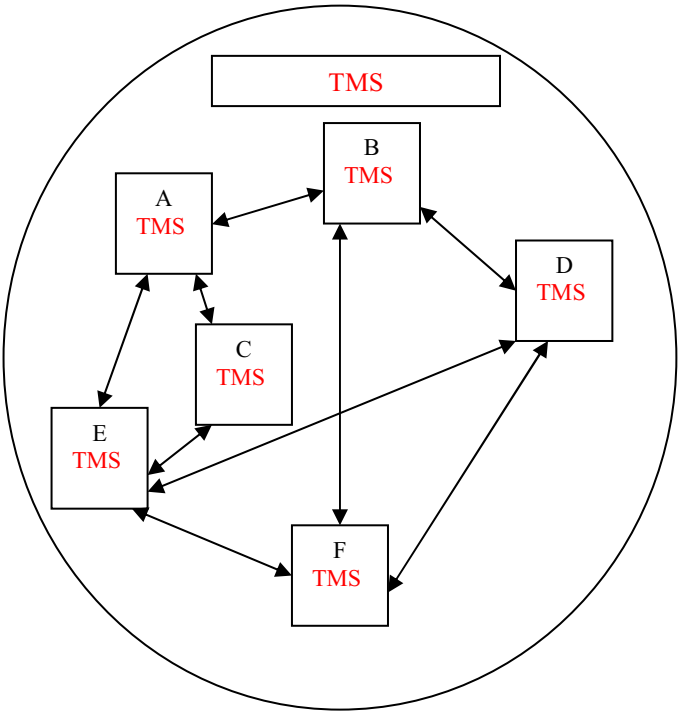
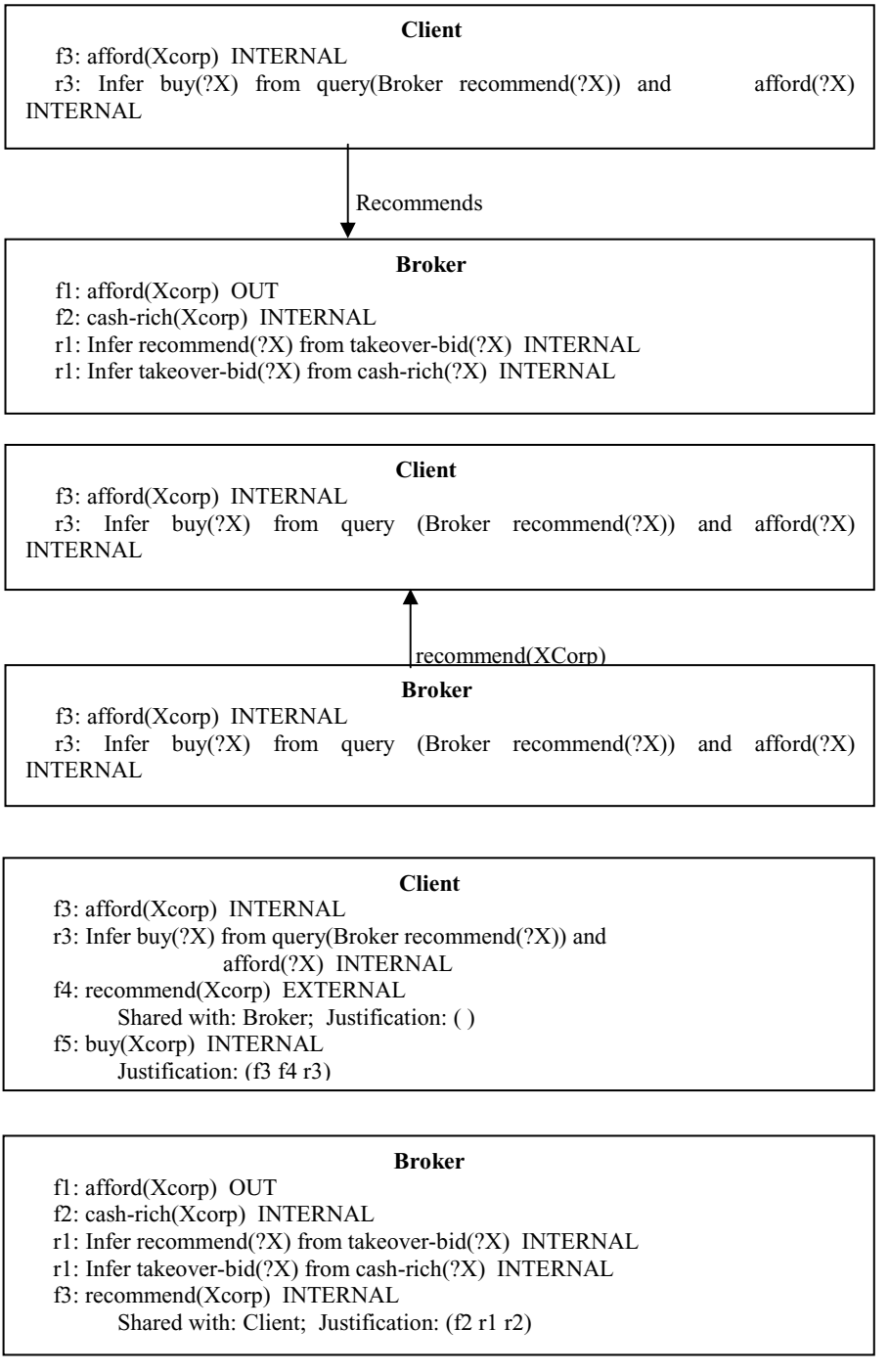


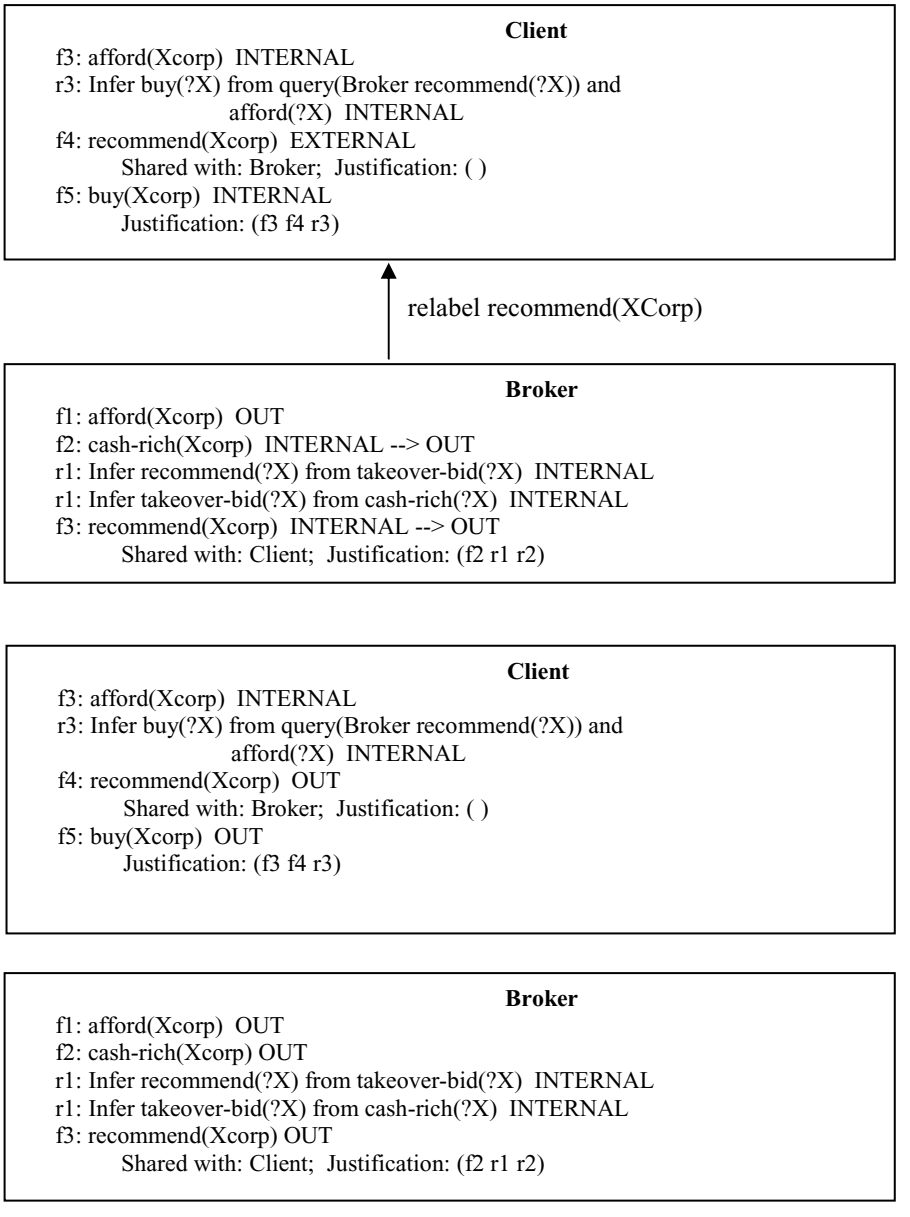
Fig. 1. A group of interacting agents

3 Truth Maintenance Systems

TMSs (Truth Maintenance Systems), also called Reason Maintenance Systems are one of the common tracking mechanisms for achieving knowledge base integrity [1]. TMSs support default reasoning and provide justification for conclusions. One of their basic uses is recognizing inconsistencies. Moreover, TMSs can remember the previously computed derivations. They also support dependency-directed backtracking [3]. There are different types of TMSs, namely Justification Based Truth Maintenance Systems, Assumption Based Truth Maintenance Systems, Logical Based Truth Maintenance Systems, etc. Henceforth, we employed a kind of justification based TMS in our work to meet our system's requirements. A justification TMS, in general, is a TMS where each proposition or assertion in it has a justification, and is labeled as IN (believed) or else, OUT (disbelieved) [1], meaning that it is not justified, or it is invalid. TMSs can be used for achieving the knowledge base integrity in a single agent system and can be used for multiagent system negotiations as well.

The following example shows how the TMS works. The example is adapted from the Service-Oriented Computing Semantics, Processes, Agents [9].


Client
f3: afford(Xcorp) INTERNAL
r3: Infer buy(?X) from query(Broker recommend(?X)) and afford(?X) INTERNAL
Recommends
Broker
f1: afford(Xcorp) OUT
f2: cash-rich(Xcorp) INTERNAL
r1: Infer recommend(?X) from takeover-bid(?X) INTERNAL
r1: Infer takeover-bid(?X) from cash-rich(?X) INTERNAL
Client
f3: afford(Xcorp) INTERNAL
r3: Infer buy(?X) from query (Broker recommend(?X)) and afford(?X) INTERNAL
recommend(XCorp)
Broker
f3: afford(Xcorp) INTERNAL
r3: Infer buy(?X) from query (Broker recommend(?X)) and afford(?X) INTERNAL
Client
f3: afford(Xcorp) INTERNAL
r3: Infer buy(?X) from query(Broker recommend(?X)) and afford(?X) INTERNAL
f4: recommend(Xcorp) EXTERNAL
Shared with: Broker; Justification: ( )
f5: buy(Xcorp) INTERNAL
Justification: (f3 f4 r3)
Broker
f1: afford(Xcorp) OUT
f2: cash-rich(Xcorp) INTERNAL
r1: Infer recommend(?X) from takeover-bid(?X) INTERNAL
r1: Infer takeover-bid(?X) from cash-rich(?X) INTERNAL
f3: recommend(Xcorp) INTERNAL
Shared with: Client; Justification: (f2 r1 r2)



4 Specifications of Our Model

In our system, we have two types of propositions:

Premises: these are the propositions sent to the agent by other agents [4]. Once an agent accepts a premise from another agent, she will always consider it to be true. Neither does the agent have access nor does it need justifications for the soundness of this premise from the sending agent once the premise is accepted by her.

Derived Propositions or just (Propositions): The derived propositions are the ones that are inferred or constructed in the agent's database based on the Premises she has and her previous knowledge [4].

We will have in our system a new type of information which will play a principal and key role in all discussions. This information will be saved in the agent's Database, and it will be considered part of the agent's previous knowledge. A set of inference rules will be hard coded and saved in the soundness Handler component. To put it differently we will take each inference rule such as A + B → C and hardcode this in the agent's database. The following table illustrates this process:

```
set op1 = operation
set operator1 = A
set operator2 = B
set operator3 = C
if ( op1 eq? +)then
{
   operaotr1 →  operator3 ( is true)
  operator2 → operator3 (is true)
}
else if(op1 eq? *)
{
    operaotr1 →  operator3 ( is false)
    operaotr2 →  operator3 ( is false)
 }
```

All future judgment for propositions will depend on this notion and the notion of justification. According to the inference rules, if a proposition is believed, then it means that there is a set of other propositions justifying it. This new justified proposition will be added to the system and will be called the descendent of the justifier's proposition. The justifying propositions are called the ancestors of this proposition. In the following discussion, we will shed more light and give more details about this notion.

5 Soundness Parameter

We posit consider that a contradiction will not be forecasted until it occurs. Upon assertion of a proposition, it is justified. After a while, there might exist a proposition with its negation in the database of an agent, leading to a contradiction. Our focus is to find ways that keep every agent perpetually consistent by not permitting any proposition and its negation to exist simultaneously. For that, we define a new *soundness* parameter, which allows us to resolve arising conflicts. This new *soundness* parameter is used to check the soundness of the derived propositions in the agent's database. We will rely on our new parameter and whether the proposition has been justified or not to resolve any conflict or contradiction.

By using our *soundness* parameter, and whether a proposition is justified or not, we will have a strong judgment in retracting any of the propositions stored in the system, lowering the chance of retracting any wrong propositions.

Soundness: is a parameter that quantifies the validity of each proposition in the agent knowledgebase. Its value is in the interval [0, 1].

This is different from the traditional labeling algorithm [6], which is used to retract some of the propositions in the agent to remove the conflict, and it is also different from the set of retraction steps proposed by John Doyle [7]. Instead, our parameter relies on the degree of support. If a proposition is found to have high support then it will not be retracted, but if it is found to have low support it may be retracted. We have developed our model and assigned a value for our *soundness* parameter for each proposition based on our model. After applying our parameter and our criteria of retraction, the conflict will be removed with a very low likelihood of choosing a wrong proposition.

For a proposition P, we assign a *soundness* value based on both the premises in the system, and the previous knowledge of the agent. This value can be one of three values:

- < 0.5 which means that its soundness is low/(very low), and hence, this preposition may be retracted.
- > 0.5 which means that its soundness is high/(very high), and hence, this preposition will not be retracted.
- $= 0.5$ which means that its soundness is neither high nor low, and hence, this proposition may or may not be retracted, depending on the situation. So, more knowledge is needed.

Before explaining how we got these values, it is important to show how the judgment is formulated. Before checking for the soundness value of a proposition, a check is performed to see whether the TMS has been able to justify this proposition or not.

After this check, the soundness value will be computed. If the proposition can be justified, then we can expect a high value of soundness, but this may not be the case.

If the proposition cannot be justified, then there are two situations to be considered. The first one is when the proposition cannot be justified because it is invalid; the other is when the proposition cannot be justified because there is no enough knowledge about this proposition, nor is there any evidence that this proposition is invalid.

The following is a description of the way the *soundness* parameter values are achieved. In order to understand the way the soundness values have been computed, we will consider the following example:

Suppose we have the following Premises in Agent's A Database:

$$A + B \rightarrow C \quad (1)$$

$$D * E \rightarrow F \quad (2)$$

If the following proposition is derived:

$$A \rightarrow C \quad (3)$$

Then from (1) and the agent's previous knowledge of inference rules, this is true and, thus, the proposition will be assigned a high soundness value, like soundness(P3) = 0.9. For this proposition we can expect that it has been justified by the TMS.
If the following proposition is derived:

$$D \rightarrow F \tag{4}$$

Then from (2) the previous knowledge of inference rules, this is invalid and hence a low value will be assigned to the soundness of this proposition, like soundness(P4) = 0.3. Similarly, for this proposition, since we know it is invalid, we can expect that this proposition has not been justified since it is not a valid proposition.

We may have a different situation where no previous knowledge is known about this proposition; the following proposition is a case in point:

$$A \rightarrow G \tag{5}$$

Here, no knowledge can be used to justify this proposition. For this kind of propositions, our system, particularly the Soundness handler (discussed below), will give a value of 0.5 soundness(P5) = 0.5 for the soundness parameter, meaning that it is unknown. We will call this kind of assertions Pending. For this proposition, we can expect that this proposition has not been justified, not because it is invalid, but mainly because no enough knowledge is available to judge this assertion. In the following, further discussion for each of the above cases will be examined.

6 Components of Our Model

Figure 2 shows a schematic diagram of the architecture of our model:

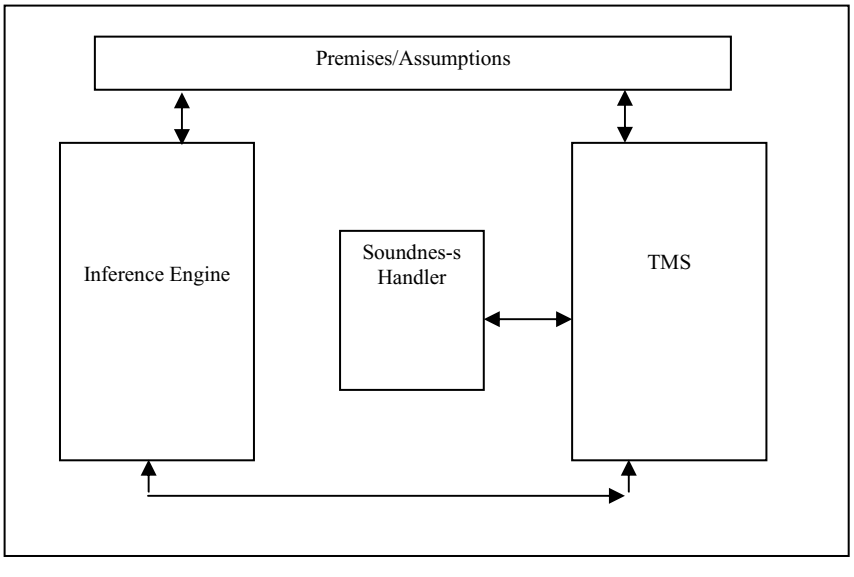


Fig. 2. Schematic outline of our model

We have the following components:

Premises/Assumptions: These are the propositions that are sent from other agents. In this model, we consider any proposition received from other agents as a premise, and therefore, it is true at all times, and does not need justification.

Inference Engine: This component creates new propositions. The creation of new propositions will depend on the agent's previous knowledge and the set of premises. However, some of the new created propositions may not depend on the previous knowledge of the agent. This does not necessarily mean that the created proposition will be true or valid. At this stage, the agent has no guarantee of soundness or validity for any proposition. Some of the propositions/assertions may violate other propositions that exist in the agent's database. There are many reasons for this. One reason may be due to the fact that new propositions fit the requirement of the current time and state, and the old ones may be out of date at this point. It is also axiomatic that new requirements arise as time progresses. Moreover, the premises are considered as eternally true propositions and they do not need justifications. These premises come from different sources or agents, and since the new derived propositions partially depend on them, some contradictions may occur in these assertions or propositions in the agent due to the disparity among different sources of the premises.

TMS: The TMS is the central component of our model. It has access to all propositions and premises. It is the component in which the propositions are justified and then labeled as *justified,* meaning that they are justified or believed; or *not-justified,* meaning that they are disbelieved. When a new proposition is created, the TMS will work to justify this proposition by relying on its previous knowledge and the set of premises the agent has. It will also control the system by detecting any contradiction that may take place. Once a contradiction happens, it tries to resolve this contradiction by retracting some of the propositions, relying on the fact whether the proposition is justified or not, and on the soundness parameter to be assigned to each proposition. In the system, every proposition and all its descendents are maintained in the agent's database. Therefore, if at any time, a contradiction happens, the system will be able to keep track of the propositions, and follow the chain of propositions or assertions from the current proposition, going back to its ancestor until getting to the set of premises. By this, the set of propositions that need to be retracted will be specified. The retraction process will be performed using Dependency- Directed Backtracking [2].

Soundness Handler: This is a novel component where each proposition receives its soundness value. A soundness value will be any value between 0 and 1 inclusive. A proposition which is found to be valid will be assigned a soundness value of more than 0.5, and will be labeled as IN. By the same token, any proposition which is found to be invalid will be assigned a soundness value of less than 0.5, and then will be labeled as OUT.

However, if a TMS fails to judge whether the proposition is valid or invalid because of not having enough information about the proposition, this proposition will be assigned a soundness value of 0.5, and will be labeled as *pending*. The *pending* proposition means that it may be relabeled later as either IN or OUT, when new information or evidence of validity or invalidity is obtained. If new evidence appears

to support this proposition, this proposition will be relabeled as IN. If any new created proposition depends on one of the *pending* propositions as its justification, the *pending* proposition or assertion will also be relabeled as IN. The new proposition will be the descendent of the relabeled proposition. However, if new evidence appears to show that some *pending* proposition is invalid, the *pending* assertion/proposition will be relabeled as OUT, and its soundness value will be lowered by a value of 0.1, making its soundness value less than 0.5.

7 Conflict Management, a Scenario

Most Truth Maintenance Systems have some way of detecting a contradiction when happens. This can be done by adding a special proposition symbol called *contradiction*. If the truth maintenance system is able to detect a contradiction then the contradiction proposition will return **yes** [3].

But how can this work in resolving any contradiction?
Suppose we have the following set of inferences:

$$P1 \rightarrow P2 \rightarrow P3 \quad (1)$$

$$\text{And} \qquad q1 \rightarrow q2 \rightarrow q3 \quad (2)$$

And suppose the following soundness values 0.8, 0.5, 0.3 are assigned to each proposition of P1, P2, and P3, respectively. And the following values 0.7, 0.6, 0.9 for q1, q2, and q3, respectively. And suppose that P3 has not been justified and q3 has been justified.

Suppose we have the situation where P3 = ~q3 which causes a contradiction in the system. Then based in the soundness values, P3 will be retracted because it has a soundness value less than 0.5 which is also less than q3's value and it is not justified, leading to the removal of the conflict from the system.

Continuing with the previous example, consider the following situations:

- If soundness(q1) > 0.5 and soundness (p1) < 0.5 then retract P1 given that q1 has been justified and p1 has not been justified.
- If soundness (q1) > 0.5 and soundness (p1) = 0.5 then retract P1
- If soundness (q1) > 0.5 and soundness (p1) > 0.5 then it is not enough to check which one is greater than the other because it is very difficult to retract both of them. Also, both propositions have been justified and labeled as IN. So, this situation needs further discussion
- If soundness (q1) < 0.5 and soundness (p1) < 0.5 we can retract any of these two proposition, but for more accuracy, more discussion is needed to make the judgment which proposition to retract.

Consider the following situation where soundness (q1) > 0.5 and soundness (p1) > 0.5, the following solutions will be considered:

- Assign a timestamp for each proposition at the moment this proposition is inferred or created. For the above situation, the timestamp for each proposition is checked. The proposition with the older timestamp will be retracted. Sometimes the newest proposition is considered the freshest one and, hence, has a better chance to be the most correct one. But this

is not often the case. So, more discussion is needed to find a better solution.

- Compute the soundness average starting from the last proposition P3 going back to the first premise that participates in constructing this proposition. So:
  - Avg(soundness (Pi)) = Pi / n
  - Avg(soundness (qi)) = qi / n
  - Compare the two averages and retract the proposition with the lowest average.

Relying on the average may not always give us the correct solution, but, on average, will give the best solution.

The same solution methods of assigning timestamps and computing the average are applied in the case that the situation where soundness (q1) <0.5 and soundness(p1) < 0.5 exists.

8 Conclusions and Future Work

In this work, we studied Truth Maintenance Systems. We introduced salient components of a TMS and then we introduced a new *soundness* parameter. By combining the TMS with our new soundness parameter, we provided a novel and resilient conflict management scheme. Soundness Parameter is used to help in deciding which propositions/assertions to retract if a contradiction occurs. These help the TMSs in reducing the chance of retracting wrong propositions, lowering the chance of making wrong judgments and, at the same time, enhance the correctness of the function of the Truth Maintenance System.

For further future work, we will be working on enhancing trust among the interacting agents, and assure the information being spread among the interacting agents. This process will be achieved by combining the notion of Truth Maintenance System with the notion of trust. We will use the TMS concepts as the driving way for achieving our goals. The TMS will guarantee the consistency of all interacting agents as well as the global consistency. Once this is achieved, the trust management scheme can be applied, leading to ways that enhance trust and assure the information.

References

1. M. Huhns and D. Bridgeland: Multiagent Truth Maintenance. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 6, Nov/Dec 1991.
2. Stuart C. Shapiro: Belief Revision and Truth Maintenance Systems: an Overview and a Proposal. Technical Report 98-10, Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY, December 1998.
3. Kenneth D. Forbus and Johan de Kleer: Building Problem Solvers, MIT Press, 1993
4. M. Barbuceanu, M.S. Fox: The Information Agent: An Infrastructure Agent Supporting Collaborative Enterprise Architectures, Proceedings of 3-rd Workshop on Enabling Technologies, Infrastructure for Collaborative Enterprises, Morgantown WV, IEEE Computer Society Press, 1994.
5. David A. McAllester: Truth Maintenance.AAAI-90, pp. 1109-1116

6. J. Kleer: A general labeling algorithm for assumption-based truth maintenance, AAAI-88, pp. 188-192, 1988.
7. J. Doyle: Truth maintenance systems for problem solving. Fifth International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, 1977.
8. J. Doyle, Reason maintenance and belief revision: Foundations versus coherence theories, in: P. Ggrdenfors (ed.), *Belief Revision,* Cambridge University Press, (1992) 29-51.
9. Munindar P. Singh, Michale N Huns, Service-Oriented Computing Semantics, Processes, Agents.

Mind Out of Programmable Matter: Exploring Unified Models of Emergent Autonomy

M. Randles, A. Taleb-Bendiab, and P. Miseldine

School of Computing and Mathematical Science, Liverpool John Moores University,
Byrom St. Liverpool, L3 3AF, UK
{cmsmrand,a.talebbendiab,cmppmise}@livjm.ac.uk

Abstract. This article advocates the need for a radical rethink of software agent technology by investigating the mechanisms through which knowledge, deliberation, action and control interact to form truly intelligent autonomous agents, be they deliberative, intentional or purely reactive automata/ particles/actors. Using sound logical formal modelling techniques, this work attempts to propose a unified model of multi-agency, which integrates and consolidates various proposed software agent models including; deliberative, cognitive, collectivist and individualist agent perspectives. In particular, the paper focuses on the formal semantics of model-based and emergent regulatory structure of autonomic self-regenerative systems, agents or particles (swarm intelligence). In addition, the paper uses our new scripting language – Neptune and associated development environment, which transforms formal requirements models of a given agency to executable code.

1 Introduction: The Current Situation

Over the last 3-4 decades, many noteworthy advances have been made extending our understanding and application of Distributed Artificial Intelligence (DAI) to develop models of reactive, proactive, deliberative and/or emotional agents. Many AI systems have been produced that perform well within the domain they have been designed for, "yet they could not learn concepts that most children know by the time they are three years old."[1]. It is the aim of AI research to make computer systems perform "intelligent" tasks to make life easier for humans. This may be in the realm of pervasive computing [2], hidden in the users' environment, or in autonomic computing [3], seeking to alleviate the burden of system management. In either case it is very useful to conceive of the tasks as being carried out by agents, which are allowed the freedoms to be reactive, proactive and deliberative [4] as a generic feature, rather than in any domain specific way. In the future it may be possible to provide a computational framework where some conscious intelligence emerges as a feature of a learning system. Unfortunately AI has suffered from grand, unfulfilled promises and unsubstantiated claims. These have mostly arisen from separate isolated areas of research that seldom interact with each other. At the most fundamental level there is a divergence of method between the top-down, centralised management approach of policy-based systems, for instance, and the bottom-up devolved control

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 65 – 73, 2006.

models, capable of exhibiting self-organisation. At a more involved level various models are used to imbue agents with intelligent-like behaviour. Researchers work on producing Artificial Immune Systems (AIS) [5], Autonomic Systems [3], Sensor Networks [6] and many other features, of a biological entity, which can be seen as possessing intelligent responses to stimuli, whether through reaction, anticipation or deliberation. These can all be realised to some extent using agent technology, in particular autonomous agent methods, but they are developed separately with no overall model of coordination. This is, in part, due to the huge variety of environmental information and potential events that can occur. A pre-defined model cannot possibly capture the requisite system varieties. Another factor is the replication of intelligent like behaviour by mimicking biological systems cannot in itself produce a consciousness capable of true intelligence [7]. Rather to produce anything resembling a conscious intelligence requires a system to evolve to a point where truly intelligent behaviour emerges. That is conscious intelligence cannot be programmed from the top-down but might emerge from the bottom-up interactions within a suitably created framework. A collective of self-regenerative and self-adaptive agents can provide just such a framework with simple specified notions of a collective integrity to define federation membership and the interactions between members. Additionally the use of concepts such as diversity, cognitive immunity, scaleable redundancy and threat reasoning, to change and adapt to circumstances in response to the variety of external or system stimuli, can be used to handle the complex variety of environmental data and exogenous actions.

2 Artificially Intelligent Agents

The exhibiting of truly intelligent behaviour requires a system to have something akin to a consciousness. Dreyfus [7] argues that it is impossible to create an artificial mind. Any attempt to program a fully-formed intelligence inevitably encounters many "conceptual inconsistencies." However this does not rule out the provision of a system framework where conscious intelligence may emerge over time. In a similar way children only appear to exhibit responses to stimuli at birth. Intelligent behaviour emerges as the child's mind learns and develops.

A system framework needs to be established that handles the complex and competing demands of environmental and social interaction and allows the emergence of notions that may lead to a collective, acting as a single entity, showing truly intelligent behaviour.

2.1 Architecture for Intelligence

To model the complex structure of an intelligent agent a blueprint can be applied to model the system as a single agent with embedded subagents conforming to the same blueprint, in a similar way to humans comprising of many interacting lower level systems [8]. In turn these subagents contain their own embedded subagents giving a fractal like structure to model the system complexity. This has been represented in the Viable Intelligent Agent Architecture (VIA-A) architecture [9], which is based on an extension of the well-established cybernetics representation the Viable System Model

(VSM) [10] combined with elements of the IRMA [11] model. The model identifies five subsystems within a running system: The effectors (S^1), coordination (S^2), management (S^3), an audit component (S^{3*}), deliberation on future scenarios (S^4) and system identity (S^5). Each of these systems also embeds its own copy of the S^1-S^5 structure to any level required. Also, alternatively to separately developed, single concept agent software systems, human agents function as a complex, yet coordinating, hybrid of all sorts of systems. There is an autonomic system that self-regulates the body; there are the five senses that provide environmental input and so on. But these do not act in isolation but overlap and coordinate in a complex way to deliver the complete autonomously acting system. In a similar way a computer system based on the VIA-A design ought to be viewed as a holistic entity where the various systems of agents and methods of control interact and no one paradigm is used to enforce a total modelling methodology throughout the blueprint or at any level of the subsystem. The overall model ought to include both a top down normative intentional approach to deliberation and a bottom up distributed control mechanism with methods such as Chance Discovery, Novelty Detection or Danger Theory [12,13] used to detect emergence via suitable partial observation conditions through which a conscious intelligence may result.

2.2 The Emergence of Intelligence

A framework, as described above, delivers a number of vital prerequisites for the emergence of intelligent behaviour.

- The system has access to real world perceptions and information. Exposure to the chaotic, complex nature of the world emphasises the necessity of developing a sophisticated method of attenuating data and amplifying actions in the world. Such a method would be conscious intelligence. The framework as described relies on subsystems interacting in suitable ways with the world. S^1 systems rely on environmental triggers for some actions whilst others result from environmental interactions propagated down from higher S-systems.
- The system is able to reason and has the capability to act on this reasoning. That is the system is able to act on the world and has reasons for this action. A system that could only absorb information but not act on it would not develop consciousness. Conversely a system that didn't need environmental interaction could not develop any intelligence to exist in the world.
- The system has the capacity for learning. It is has the ability to improve itself through feedback and sensing systems. This however is radically different to how human learning takes place. The emergence of a conscious intelligence allows the forming of casual relationships. At present computer memory functions quite differently to human memory. A computer memory holds all information in perfect detail and is thus flooded with many insignificant details. The emergent conscious memory would be able to hold fuzzy memories capable of assembling necessary relationships as necessary, allowing much faster responses.
- The system is able to propagate itself. Self-regeneration is available to enhance the viability of the system. It is likely that as consciousness emerged such

methods would evolve to be a less mechanistic and artificial. An empathy and social ability would follow as truly intelligent behaviour.

- The system displays all the facets of complexity mathematically necessary for emergence to occur [14]. That is the system comprises of mutually influencing components, different system components enhance other components, the system takes from the world dissipating the resulting increase in entropy back into the environment and the components are subject to random world events.

Thus the next section proposes a hybrid model that stresses the framework where a central module has an evaluating role to proscribe unsafe or detrimental behaviour. Importantly however there is also the specification of low level interactions between agents that can coalesce in emergent structures that collectively exhibit intelligence.

3 A Hybrid Agent Design: Top Down Versus Bottom Up

A blueprint, such as the VIA-A architecture, provides the requisite means of reasoning about complexity in the system. Additionally intelligence has a chance to emerge from the system. It is proposed to model the various systems as hybrid centralised/decentralised control agents. This is achieved through the use of a centralised deliberative module, acting as an observer, with control abstracted out through environmental semiotics (Stigmergy [15]) and self-organisation through the emerging swarm intelligence. By studying such models, self-organisation of computer systems can be arranged, where centralised governance is replaced with control distributed across all the participating agents with the possibility of emergent intelligence. Such systems are better modelled by a bottom-up approach where the basic agents and their interactions in the system, and the environment, are defined. In this way a federated intelligent behaviour is not prescribed but rather emerges from the system. So although, as a software entity, the system controller is still extant, acting as a top down observer, the intelligence is actually abstracted by distributed knowledge. The observer views and appreciates emerging patterns of organised behaviour, unknown to the low level collective participants. In this way the system control is divested from a central controller to the active participants within the process. This leaves the central controller with a less complex (easier implemented) monitoring observer's role in the system, yet retaining deliberative capabilities based on its normative state.

Consciousness represents a way to organise and use information that can emerge as a collective facet of agent federations. The central observer acts as a controlling influence on the intelligence that can emerge from bottom-up interactions in the agent system. Searle states that "consciousness serves to organise a set of relationships between the organism and both its environment and its own states" [16]. Thus consciousness represents an awareness of self existence and the world around, with the ability to react in a consistent interested manner to it.

Obviously, at present, there is no conscious intelligence in computer systems. However the specification of interaction frameworks, where very large numbers of simple agents can form teams, collectives or swarms may approach a conscious entity. The mass flow of information through the federation and environment, may give intelligence that is not attainable through rudimentary programming methods.

4 Formalising Knowledge and Control

In order to provide the formalism that allows analyses of the mechanisms by which intelligence may emerge it is necessary to consider the emergent nature of mass swarm-like behaviour. The detection of novelty and emergent behaviour is very difficult. What is sought is a theoretical and implementation method for autonomous software agents to sense, reason and act in transitional, only partially observable, non-predictable environments where the local interactions of the participants abstract out the need for centralised control and exhibit intelligence. This leaves only detection to promote safe behaviour and proscribe undesirable emerging properties as a central (observer) role. Thus agents' beliefs and the ensuing actions prompted by these beliefs are paramount. These beliefs are best represented as logical sentences [17] that condition behaviour. This behaviour affects other members of the collective and in such a way it is hoped to provide a formal and computational account of deliberation leading to action in a swarm type system (Swarm Calculus). Thus there is an emphasis on beliefs and how they condition individual behaviour, to influence group behaviour. In this way intelligence emerging from the system is conditioned by the beliefs inherent in the mass system as held by the vast collection of participating agents. This, in turn, means that logical consequences of emergent behaviour, whether inferred or detected, can be ascertained, evaluated and utilised within the system. As an example, using dynamic logic with the usual modal operators □ meaning "it is necessary that" and ◊ meaning "it is possible that", consider a system where the number of agents, in a team, needs to be maintained above a certain number N. A mutual goal, p, for a team (T) can be stated as:

$$G(T,p) \Leftrightarrow B(T,\neg p) \wedge G(T,p) \wedge [UNTIL[B(T,p) \vee B(T,\Box\neg p)]$$
$$LTG(T,p)$$

where B is the modal belief operator [18] and LTG is a lesser team goal that the team mutually believes that all the members have a team commitment to p

So we have

$$\models G(T,numberofmembers(T)>=N)$$

From this we can infer, as a logical consequence, that team members as individuals have a commitment to maintain the number of colleagues above a specified level so that when one believes that the number of fellow members is less than required and believes that this is not mutually believed by the team and believes it is not impossible to establish mutual belief for the team then it has an individual commitment to bring about this mutual belief.

i.e. $\models G(T,p) \Rightarrow \forall t \in T\ [B(t,\neg p \wedge \neg C(T,\neg p)) \wedge$
$\neg B(t, \Box\neg C(T,\neg p)) \Rightarrow G(t,C(T,\neg p) \wedge B(t,\neg p))]$

where p is *,numberofmembers(T)>=N* and C is the common belief operator [18] which is proven by assuming:

$$\forall t \in T\ [B(t,\neg p) \wedge \neg C(T,\neg p)) \wedge \neg B(t, \Box\neg C(T,\neg p))$$

Then from the definition of team goal:

$$B(t, \neg p) \wedge \neg C(T, \neg p) \Rightarrow G(t, C(T, \neg p))$$

and G(t, C(T,¬p)) is satisfied because if one member of the team does not believe there is mutual belief then there is no mutual belief. So since the consequent of an implication must remain true until the antecedent or the implication statement becomes false

G(t, C(T,¬p)) holds until

$$t \in T\ B(t, \neg p \wedge \neg C(T, \neg p)) \wedge \neg B(t, \Box \neg C(T, \neg p)) \text{ doesn't;}$$

that is until $\neg B(t, \neg p) \vee C(T, \neg P) \vee B(t, \Box \neg C(T, \neg p))$ is true.

So all the conjuncts in the definition of $G(t, C(T, \neg p) \wedge B(t, \neg p))$ are satisfied proving the result. Similar results follow as a matter of logical consequence. Knowledge producing actions, whether sensory, inferential or discovered, lead to consequences for the system that can be established via this formal representation.

5 The Story So Far: Calculus to Code

The above example shows one simple rule of interaction. It is an important rule in maintaining a specified number of agents in a team, as each collective will require some minimum number of entities to function optimally. However there will be many, many such imperatives from which emergent behaviour will arise from association with the team, the world and other systems. Some imperatives will follow as logical consequence to a minimal set of imperatives whilst the specification allows emergent intelligence.

5.1 Formalisation

As can be seen from the brief example above; logic based formalism works well in using system knowledge to extract necessarily true system properties as logical consequences of the specification. The Situation Calculus [19] further provides knowledge capturing and deliberation techniques to more than adequately reason about complex, dynamical agent systems. It is a first order predicate calculus with some second order features giving excellent expressiveness with good reasoning methods. In this way the emergent logical consequences of the specification can be considered and verified whilst emergent behaviour arising from the inevitable incompleteness of the specification can be handled via a dynamic observer monitoring for novel system aspects and consciousness intelligence. The further introduction of stochastic actions to the Situation Calculus representation allows extraneous and natural events to be included. The Stochastic Situation Calculus therefore provides a mapping from knowledge to stochastic actions, including environmental actions, based both on sensing mechanisms and logical reasoning. As a representational tool it does not require an explicit enumeration of the state space, thus permitting unknown situations. In this way it can be used as a Swarm Calculus, for a bottom-up approach, and as a behavioural deliberative model in norm based systems, for a top-down, model-driven approaches. Its use here is to provide a unifying design model, within the recursively embedded VIA-A structure, to represent composite

swarm-based/deliberative agents and agent systems that have the capacity to regenerate themselves. This encourages fault tolerance and redundancy through diversity, cognitive immunity by adapting and generating agents, to learn from past failures, and deliberation for adapting to threats. It specifies the interactions of the huge systems that may lead to some form of consciousness unavailable from static program structures.

5.2 Implementation

A new scripting language, based on logical statements, has been developed that is very useful in taking the logical sentences and transforming them into run time adaptable and addressable objects. This language, Neptune [20] thus provides the means to realise, in practice, the desirable system properties that are evident in the logical specification. Thus agent systems exhibiting human-like behaviour can be constructed from autonomous subagents acting in teams to provide distributed control following as logical consequences from the specifications. Additionally chance discovery can be performed by a "stripped down" system controller assessing emergence and providing a norm based deliberative agent connotation. In this way as this work progresses it is hoped to provide a way for consciousness and thus intelligence to emerge.

5.3 An Example Application: Dynamic Physical Rendering

Dynamic Physical Rendering involves the creation of physical moving 3-dimensional replicas of objects. This involves using captured 3-D images and rendering them through programmable matter called claytronics [21]. The basic unit of claytronics is a *catom*. Replicas are then produced from a collective of catoms. Catoms have four capabilities: computation, power, movement and communications. It is clear how the work described in this paper is applicable to claytronics. The rendered physical representation will be a collective of many catoms. These catoms will be required to interact and be controlled in a manner that allows emergent properties to appear yet will be subject to some central authority to shape the entity required. So, for instance, an entity will require a minimum number of collected catoms and behaviour, as shown in the example, will follow from this.

Each catom has its own computational power. So suppose there is a simple rule (stated in the situation calculus) of interaction that the catom is required to be maintained at a background colour with:

$$\text{colour}(\text{do}(a,s))=C \Leftrightarrow [\text{colour}(s)=C \wedge a \neq \text{changeColour}(C')] \vee a=\text{changeColour}(C)$$

$$\text{poss}(\text{changeColour}(C),s) \Rightarrow \text{backgroundColour}(s)=C$$

Thus we can supply a simple logical model of the interactions of the catom with its environment for use in the overall large specification that determines collective association and behaviour.

In turn this rule may be implemented through Neptune to provide an adaptable software object attached to the catom. i.e.

```
situation sChangeColour to Catom C
{
   when (changed c.Colour)
}

action aChangeColour situation sChangeColour to Catom C
{
   sChangeColour.C.Colour = C.Colour;
}

prediction pChangeColour to situation aChangeColour
{
   aChangeColour.sChangeColour.C.Colour =
aChangeColour.C.Colour;
}
```

In this way a method of changing colour for a simple coloured catom is specified, which is part of the catom and allows interaction with the environment. Additionally this software object is adaptable should this be necessary through the runtime governance of the system.

6 Conclusion

Work has already been completed on an agent-norm based approach to system self-governance. This has been implemented as a medical decision support system exhibiting deliberation on knowledge for both application and system level control. It is implemented in a grid computing environment using autonomous agents in a top-down system controlled norm-based application. The work on Swarm Calculus, towards a means of producing some notion of consciousness, is more recent. The programming model is seeking to make use of a calculus to code design so that the logically developed methods of modelling swarming and emergent behaviour can have a verifiably correct implementation into code and display self aware intelligence. This forms part of the ongoing work, on many fronts, to develop a unifying model of an intelligent, generic agent. At a nanotech level such work has implications for the control of small components (agents) to be used for the construction of larger agents (e.g rendering a claytronic entity). It is hoped, by this holistic approach to bridge the gap between the learning ability of the human mind and that of software agents by modelling a software system as an agent consisting of a complex arrangement of adaptable agent subsystems utilising different organisational and deliberative schemes within a unified representational structure to ultimately display true intelligence.

References

1. M. Minsky, “Building Intelligent Machines”, Speech at Boston University, May 16, 2003, http://wiredvig.wired.com/news/technology/0,1282,58714,00.html
2. A. Schmidt, K. van Laerhoven, “How to Build Smart Appliances” *IEEE Personal Communications,* Vol.8(4): 66-71, 2001

3. P. Horn, “Autonomic Computing: IBM Perspective on the State of Information Technology”, *IBM T.J. Watson Labs,* New York, 2001, http://www.research.ibm.com/autonomic
4. N. Jennings, M. Wooldridge, “Agent-Oriented Software Engineering” *Handbook of Agent Technology,* ed. J. Bradshaw, AAAI/MIT Press, Cambridge MA, 2000
5. M. Burgess ‘Computer Immunology’ *In Proceeding of 12th Systems Administration Conference* (LISA’98), 283-298
6. A. Lim "Distributed Services for Information Dissemination in Self-Organizing Sensor Networks," ***Special Issue on*** *Distributed Sensor Networks for Real-Time Systems with Adaptive Reconfiguration, Journal of Franklin Institute*, Elsevier Science Publisher, Vol. 338, 2001, pp. 707-727
7. H. L. Dreyfus, “*What Computers Still Can't Do: A Critique of Artificial Reason.*” Cambridge: The MIT Press, 1993.
8. M. Randles, A. Taleb-Bendiab, P. Miseldine, A. Laws “Adjustable Deliberation for Self-Managing Systems”, In proc. *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based* Systems (ECBS 2005): 449-456, Washington MD, April 2005
9. A.G. Laws, A. Taleb-Bendiab, S. J. Wade, D. Reilly (2001) ‘From Wetware to Software: A Cybernetic Perspective of Self-adaptive Software’. IWSAS 2001:257-280
10. S. Beer, “The Viable System Model: its provenance, development, methodology and pathology”, in R. Espejo and R. Harnden, *The Viable System Model: Interpretations and Applications of Stafford Beer's VSM*, John Wiley and Sons Ltd., Great Britain, 1989
11. M.E. Bratman, , D.J. Israel, and M.E. Pollack, *Plans and Resource-Bounded Practical Reasoning,* Computational Intelligence, Vol. **4**, No. 4, pp. 349-355, 1988.
12. Y. Ohsawa (editor), “Proceeding of the 1st International Workshop on Chance Discovery”, *Japanese Society for Artificial Intelligence,* 2001
13. M. Randles, A. Taleb-Bendiab, P. Miseldine,” Using Stochastic Situation Calculus to Formalise Danger Signals for Autonomic Computing” Proceedings of PGNet 2005, Liverpool John Moores University, 2005
14. P. Glansdorff, I. Prigogine “Thermodynamic Study of Structure, Stability, and Fluctuations” Wiley New York 1978
15. Hadeli, P. Valckenaers, M. Kollingbaum, H. van Brussel, “Multi-agent Coordination and Control using Stigmergy” *Computers in Industry* 53(1): 75-96, 2004
16. J.R. Searle *The Rediscovery of the Mind.* The MIT Press, Cambridge:1992.
17. B.C. Smith “Reflection and Semantics in a Procedural Language” *PhD. Thesis,* MIT, Cambridge Mass., 1982
18. J.Y. Halpern, Y. Moses (1990) “*Knowledge and Common Knowledge in a Distributed Environment*” Journal of the ACM, 37(3) pp: 549-587
19. H. J. Levesque, F. Pirri and R. Reiter (1998) ‘Foundations for the Situation Calculus’. Linköping Electronic Articles in Computer and Information Science, Vol. 3(1998): nr 18. http://www.ep.liu.se/ea/cis/1998/018/
20. P.Miseldine, A. Taleb-Bendiab “Rainbow: An Approach to Facilitate Restorative Functionality within Distributed Autonomic Systems” Proceedings of PGNet 2005, Liverpool John Moores University, 2005
21. S. Goldstein, “Claytronics: A scalable basis for future robots,” *Robosphere 2004*, Moffett Field, CA, November 2004,

Characterizing Environmental Information for Monitoring Agents

Albert Esterline[1], Bhanu Gandluri[2], and Mannur Sundaresan[2]

[1] Department of Computer Science
[2] Department of Mechanical and Chemical Engineering
North Carolina A&T State University, Greensboro, NC
esterlin@ncat.edu, bpgandlu@ncat.edu, mannur@ncat.edu

Abstract. A multiagent architecture for vehicle and structural health monitoring is proposed. A prototype using this architecture was developed using JADE. Critical aspects of the design were verified using the SPIN model checker. The tasks in our framework are related to data-fusion levels and Gibson's realist position on direct perception of objects and affordances. We show how a system consisting of a multiagent system along with the monitored platform exhibits behavior at several levels, from the physics of acoustic emissions to inter-agent conversations expressing desires and beliefs. Communication, perception of public events, and system design conspire to provide the common knowledge needed to coordinate diagnostic tasks.

1 Introduction

The rising costs of current maintenance paradigms for expensive, safety-critical, or mission-critical structures motivates the search for health monitoring systems that can determine the status of their platforms in near real time. An architecture for such a system must address highly dynamic relations among computational components. We propose a multiagent architecture for vehicle and structural health monitoring. The next section presents background on this architecture. The need and requirements for health monitoring are outlined, the general nature of multiagent systems is discussed, and applications of agent technology to vehicle health monitoring are reviewed. Since our prototype system uses JADE, it and the FIPA reference architecture are outlined as well as the contract net protocol, which provides flexibility while respecting agent autonomy. Section 3 presents our architecture in terms of the kinds of agents and their interactions. Since the behavior of non-trivial concurrent systems is too complex and uncontrollable to be thoroughly tested, formal methods have been developed to verify their designs. One such method, model checking, is used in the current study and is discussed in section 4. Section 5 discusses the implementation of our prototype health monitoring system. In section 6, we relate the tasks in our framework to data-fusion levels and Gibson's realist position on direct perception of objects and affordances. We show how a system consisting of a multiagent

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 74–85, 2006.

system and the monitored platform exhibits behavior at several levels. Communication, perception of public events, and system design conspire to provide the common knowledge needed to coordinate diagnostic tasks. Section 7 concludes and suggests future work.

2 Background

2.1 Vehicle Health Monitoring Systems

Restoration maintenance involves replacing a part only when it has completely failed, but this practice leads to overall deterioration of system performance. Use- or schedule-based maintenance is preventative maintenance done after certain hours of service or a period of time [1]. Worst case scenarios are taken into account and there is wastage of useful components [2]. These practices incur huge costs involving maintenance of inventory, quality checking of removed parts, and labor [3]. Such costs are driving the research towards a system whose status can be accurately determined in real time. The system must identify any deterioration in the structure before it becomes catastrophic [4]. In particular, maintenance is based on the condition of the structure rather than the hours of service. The shift from reliability-centered to condition-based maintenance has led to structural health monitoring, where a structure is diagnosed continuously and remedial action is taken as needed. Safety and reliability are added to the structure because of an efficient and reliable health monitoring system [1].

For diagnosis, certain other tasks have to be done first: data collection, signal processing, feature extraction and classification [1]. Various techniques are used for each of these tasks. Data collection is done by various kinds of sensors and data acquisition equipment. Signal processing is done by various hardware- and software-based techniques. Feature extraction and classification use various analytical and non-analytical techniques. As these systems evolve, the coordination of the sensing, communication, feature extraction, and diagnosis will be critical [5]. An architecture that addresses this diversity of sensors and techniques as well as the huge amounts of data is needed.

2.2 Multiagent Systems for Vehicle Health Management

Wooldridge [6] defines an agent as a computer system capable of autonomous action that meets its design objective in the environment in which it is situated. An intelligent agent in particular is capable of flexible autonomous action. The characteristics of multiagent environments identified by Huhns and Stephens [7] are that they "provide an infrastructure specifying communication and interaction protocols," they are typically open, and the agents are autonomous and distributed. Huhns and Stephens also see coordination as critical and, in the case of self-interested agents (where negotiation is involved), as a major challenge.

Some studies on structural health monitoring mention agents as an immediate future trends [4]. Agents have been developed for patient monitoring [8], airport

gate assignment [9], and machinery health monitoring [10]. Such agents are decision modules that use knowledge-bases, neural networks, and similar techniques. Agents are used for structural health monitoring in Lockheed Martin's MENSA architecture [11]. While, however, there is some cooperation among these agents, the interaction among them is not principled and control flow is rigid. To enjoy advantages of multiagent systems such as flexibility and fault-tolerance, a team of agents must use a protocol to structure their activity in a collaborative and flexible way [5].

2.3 Contract Net Protocol

The contract net protocol [12] is a protocol for high-level distributed problem solving whose efficiency has been demonstrated in flexible manufacturing [13] and other fields. An agent has a task that it decomposes into subtasks for which it becomes the manager. For each subtask, the manager broadcasts a task announcement. Each agent evaluates the task description in the announcement; if it wants the task, it sends a bid to the manager. The manager uses the descriptions in the bids to rank them and sends an award to the agent with the winning bid, now known as the contractor. When the contractor completes the subtask, it sends a final report to the manager. When all final reports are in, the manager combines the results to give the result for the overall task. Since tasks themselves may have subtasks, a given agent may be both a contractor (of one task) and a manager (of another). This gives rise to an ad hoc control hierarchy structured to perform the top-level task. Manager-contractor communication that occurs between sending the award and returning the report allows the contract net protocol to be used in two complementary ways: in initialization, the protocol is used to select agents to be responsible for subtasks in an ongoing task; in operation, the protocol allows agents performing ongoing tasks to handle specific problems on the fly. As another enhancement, if a set of potential contractors is narrowed in advance, focused addressing rather than broadcasting may be used.

2.4 FIPA Reference Architecture and JADE

FIPA (Foundation for Intelligent Physical Agents) [14] is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. The FIPA reference architecture requires an agent management system, a directory facilitator, and a message transport system. A FIPA ACL (Agent Communication Language) message contains one or more parameters, such as the performative, content, ontology, and content language. JADE [15] is an open source tool to develop multiagent systems using an API provided through Java classes. It provides the runtime environment for the agents, is easily distributed across several machines, and adheres to FIPA standards. In particular, it provides classes to implement the contract net protocol. JMatLink [16] is open source software that provides an API for invoking Matlab commands from Java applications. All Matlab programs used in our prototype (for example, neural networks) are accessed from JADE agents via wrappers that we have developed.

3 Architecture

Our multiagent framework is pictured in Figure 1. None of the implementations we envision involves mobile agents, and the fact that agents may be associated with locations on the monitored platform does not force any particular distributed design. There are system agents and data-management agents, which provide services but generally do not participate in the contract net protocol. Supervisor agents are bound to one or more sensors and announce diagnostic tasks when they detect suspicious values. Monitor agents coordinate the diag-

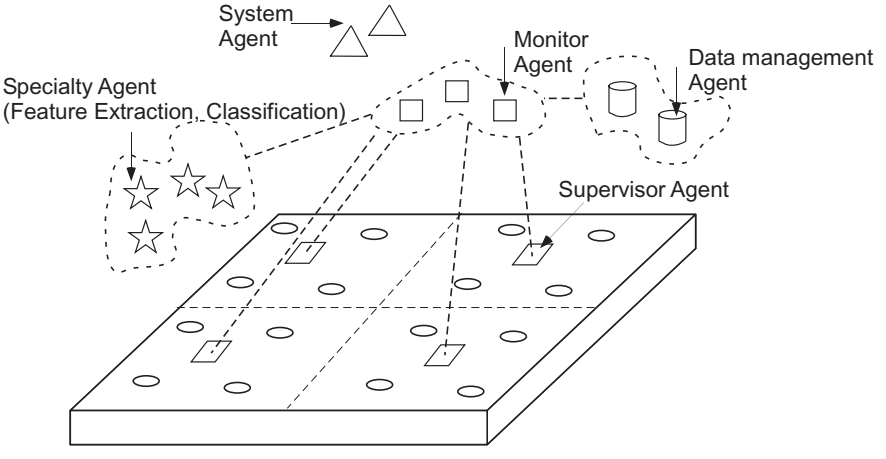


Fig. 1. Line sketch of Vehicle Health Monitoring System

nosis tasks. By participating in the contract net protocol, a monitor agent may become responsible for several supervisor agents. Other monitors may supervise the monitors that directly supervise the supervisor agents, and so on, resulting in a hierarchical configuration. Monitors focus on regions that warrant more attention and refocus as the situation evolves. A supervisor agent announces a diagnosis task to all monitors above it in the hierarchy. A specialty agent wraps a procedure for a specific task within an agent interface. In the current prototype, specialty agents include feature-extraction agents, classification agent, and location agents. In the applications we address, a specialty agent should include at least three things in its bids: an estimate of the reliability of its result, an estimate of the time it would take to produce the result, and an estimate of the bandwidth needed to produce the result. The monitor can use these estimates to make awards in a way sensitive to the current state of the system and the task at hand.

4 Model Checking

Concurrent systems present problems for testing since the behaviors of the separate threads can be interleaved in many ways, and interactions between threads

can cause the effects of inputs to cascade in complex ways. A tester's ability to exercise the behavior of concurrent systems is thus limited. The alternative to testing a system is formal verification of its design. Properties required to hold of the system (its specification) are formulated in an appropriate formalism, and the design, a blueprint for generating the system's behavior , is formulated in a less abstract notation. Then one shows rigorously that the design does not violate the specification. One such method is model checking, where the specification properties are expressed in a temporal logic and the design is expressed as one or more finite state automata. One checks that the automata, taken as an interpretation of the properties, form a model for those properties, i.e., an interpretation in which all are true. The first subsection here describes the SPIN model checker [17], and the remaining subsections describe two abstract designs we verified with SPIN along with the specifications used.

4.1 The SPIN Model Checker

SPIN's design language is PROMELA, which has asynchronous processes (threads of control), buffered message channels, and structured data. There are two ways for processes to synchronize. One process may block on an expression until it becomes true because of the action of another process. And, when one process attempts to input from a channel with a zero-length buffer (a rendezvous channel), it must wait until another process outputs on that channel; likewise, output on a rendezvous channel must wait for input. PROMELA semantics involves non-determinism at both the statement and process-scheduling levels and there is no notion of time. So design models avoid implementation detail and focus on the assumptions made within each module about interaction with others. This abstract nature is necessary since the verification procedure must check each state to verify the specification properties, and abstraction defeats complexity by drastically reducing the number of states.

Correctness claims can be inserted into a PROMELA model. For example, a *never* claim checks system properties after each statement execution for behavior that should never occur. A more perspicuous way to express SPIN specifications is with Linear Time Temporal Logic (LTL), which has two kinds of formulas: state formulas and LTL formulas. The truth value of a state formula is determined at an extended system state, which is a control state with a unique combination of values for the system variables. An LTL formula, in contrast, is evaluated only for complete runs. A state formula contains no LTL operators while an LTL formula contains state formulas and LTL operators. Regarding the operators, $p\ U\ q$ is true if p is true until q becomes true (after which p may or may not be true), $\Diamond p$ is true if p is eventually true (true now or sometime in the future), $\Box p$ is true if p is always true (true now and at all times in the future), and Xp is true if p is true at the next time. To show that a property is not violated, negate its formula. SPIN translates the LTL formula into a *never* claim that is included with the PROMELA system model, and the automata are combined. SPIN then tries to find runs satisfying the LTL formula, that is, it tries to find a falsifying behavior for the original property.

4.2 Verifying Concurrent Activity of Specialty Agents

We assume that the contract net protocol and FIPA infrastructure behave correctly and model synchronization directly between agents. Two PROMELA models were constructed. The first, called Operation, addressed the operation of monitor agents. A second, called Initialization, addressed hierarchy issues related to the establishment of a hierarchy. The intended behavior for Operation is depicted in Figure 2. We consider only one monitor agent and one supervisor agent. An event detected by the supervisor agent always results in an award to the monitor agent. We assume that the monitor agent finishes classifying and locating an event before the next one is detected. When the monitor agent finishes with one event, it waits for the next, and this carries on indefinitely. The monitor agent must manage four subtasks: feature extraction for classification, feature extraction for location, classification, and location. In this model, there is only one agent for each of the feature-extraction tasks, but there are two agents each for classification and location. Thus, the selection afforded by the contract net protocol comes into play only for deciding between the classification agents and for deciding between the location agents. The symmetry of the situation does not warrant introducing further possibilities. We want an abstract design with

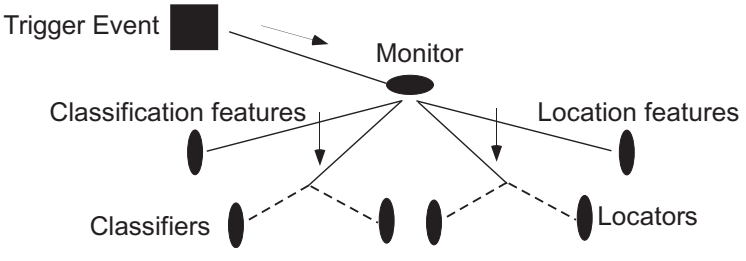


Fig. 2. Verifying Concurrent Activity of Specialty Agents

several properties. First of all, diagnosis should complete indefinitely often. Also, whenever the extraction of features for classification is finished, classification itself should eventually finish. Finally, if classification has not been completed, then it should not complete until feature extraction for it has completed. We can express these specifications in a mixture of LTL and English as follows:

$\Box \Diamond$ *Diagnosis complete*,
$\Box$ (*Feature extraction done* $\rightarrow \Diamond$ *Classification done*),
$\Box$ (! *Classification done* $\rightarrow$ (! *Classification done U feature extraction done*))

The specifications for classification and the feature-extraction tasks it depends on could be repeated for location, but symmetry makes one case representative.

The PROMELA model has ten processes representing eight agents; two dispatch processes represent aspects of the behavior of the monitor agent. All the processes execute infinite loops. Sequential constraints on the executions

of processes are mostly established with boolean variables used as guards. When a process finishes its work for the current event, it sets its controlling variable to false and sets the variables for its one or more sequels to true. To express the above specifications in a form that can be used by SPIN, we must replace the English expressions with expressions over PROMELA variables. Variable *diagend* is initially true and becomes true when the diagnosis of an event finishes; it enables the supervisor process, which sets it to false. $finish1$ is set to true by the feature extraction process for classification; it enables classification and is set to false by the monitor just before it enables classification. Finally, *done*1 is set to true by the dispatch process for classification and is set to false by the monitor just before it waits for the next event. The specifications, then, are

$\Box \Diamond$ *diagend*,
$\Box$ ($finish1 \rightarrow \Diamond$ *done*1),
$\Box$ (! *done*1 $\rightarrow$ (! *done*1 U $finish1$)).

4.3 Establishing a Monitor Hierarchy

Regarding Initiation, when a monitor agent gets a task from a supervisor agent, it may pass on the task to another monitor agent that handles a higher task. The communication in the contract net protocol and the tasks of the monitor agents are abstracted out to capture just the establishment of the hierarchy and ensuring that the hierarchy is maintained. Following the three cases in Figure 3 from left to right, initially the supervisor has a task that it announces to three monitor agents. One monitor is awarded the task, and it in turn announces the task to the remaining two monitors, one of which is awarded the task. One monitor is free, left out of the hierarchy. We want to guarantee that the hierarchy persists. One specified property is that, if the hierarchy is established, then the free monitor never gets the award. Another is that, whenever the hierarchy is established, then the top monitor eventually is awarded the task. We can express these properties in English as:

Hierarchy set $\rightarrow \Box$! *Free agent gets award*,
$\Box$ (*Hierarchy set* $\rightarrow \Diamond$ *Parent monitor handles the task*).

The PROMELA model consists of four processes, each representing an agent. One is the supervisor. There are three monitors, all instances of the same process

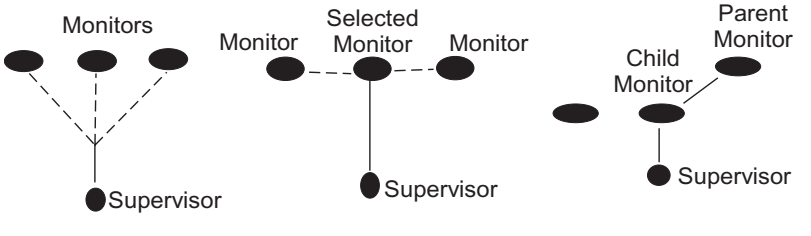


Fig. 3. Establishing a Monitor Hierarchy

type (with the same code). All processes execute infinite loops. Most of the synchronization is achieved with rendezvous channels. The process identifiers (pids) of processes are used for identification. In the chain of awards that sets up the hierarchy, monitor processes pass their pids with their reports, which allows them to be identified for subsequent awards. Selection of monitor process for award is non-deterministic during setup. Several global variables are maintained for model checking. Boolean variable *sbsq* indicates the beginning of any diagnostic task except the first. It is initially false, is set to true each time but the first that the supervisor executes, and is set to false when the child monitor receives an award. Variable $freepid$ is set to the pid of the free monitor when it receives a reject during setup, and *toppid* is set to the pid of the parent monitor during setup. Finally, *curpid* is set to the pid of the parent monitor when it receives a reward after setup. With these variables, we can now express the above LTL specifications as: $sbsq \rightarrow \Box\, !\, p$ and $\Box\, (sbsq \rightarrow \Diamond\, q\,)$, where p is defined to be $curpid == freepid$ and q is defined to be $curpid == toppid$.

5 Prototype

We have implemented a prototype of our architecture using JADE, Matlab, and JMatLink. It uses acoustic emission (AE) signals to classify events related to fatigue crack growth in materials. It has seven agents: a supervisor agent, two feature extraction agents, two classification agents, and two monitor agents. Current work is adding location agents and the necessary additional feature-extraction agents. Various parameters (features) of AE signals are used to classify their sources. AE signals from mode I fatigue crack growth normal to the direction of the load axis in fabric composite materials as well as modes I and II delamination growths are considered. Simple coupon specimens of glass fabric/epoxy composite laminate are used to generate the different types of AE signals [18]. Specimens are loaded in an MTS (material testing system) machine and the AE signals are recorded with a digital oscilloscope. The winning feature-extraction agent runs a Matlab script on the waveforms thus obtained to extract eight features, namely, symmetric amplitude and duration and asymmetric amplitude and duration for both shear and normal sensors. Two different feature-extraction agents are obtained by different instantiations of the feature extraction agent class. The values of these features are supplied to the winning classification agent, which runs a neural network for which supervised training was used. Classification identifies one of the three failure modes of the composite material. Two different classification agents were obtained by using different sets of training data.

Concerning the overall structure and operation of the prototype, each agent registers itself with the Directory Facilitator of the JADE runtime environment. An agent looking for a service finds the agents providing it, allowing the contract net protocol to use focused addressing. In the prototype, a potential contractor responds to an announcement with a randomly-generated number thought of as its estimate of the time to perform the announced task. A diagnosis task is

initiated when the user provides the supervisor agent with an ordinal number identifying the data set for the simulated fault; this action is the triggering event for the supervisor agent. The supervisor agent then selects a monitor agent through the contract net protocol. With the award, the supervisor agent passes to the monitor agent the ordinal. The monitor agent then announces a feature-extraction task, and the selected feature extraction agent is passed the ordinal of the data set. The feature extraction agent runs a Matlab script on the data set and stores the feature values in a file whose name is reported back to the monitor agent, which then announces a classification task. The selected classification agent connects to Matlab, inputs the file with feature values to the trained neural network, and reports the classification back to the monitor agent, which reports it to the user.

6 Agent Ecology and Societies

A health monitoring system benefits from sensors distributed over the structure to whose data we can apply data fusion, "the process of combining data or information to estimate or predict entity states [19]". The OODA model was an early data-fusion model that divided the fusion process into four stages: observe, orient, decide, and act. The waterfall model represented a bottom-up tree from sensor observations to decision making. Both were too rigid and circumscribed [20] and have been largely superseded by the JDL Data Fusion Model [19], a functional model that does not dictate a processing sequence. Instead, it recognizes five functional levels:

- Level 0 (Signal Refinement) involves sub-object data assessment that addresses pixel/signal-level data.
- Level 1 (Object Refinement) estimates and predicts entity states and includes tracking and classifying objects.
- Level 2 (Situation Refinement) infers relations among entities
- Level 3 (Threat Refinement) assesses the potential impact of the situation based on a priori knowledge and prognosis.
- Level 4 (Process Refinement) manages resources to give a form of adaptive data acquisition and processing to support mission objectives.

In health monitoring, it is often more natural to speak of events rather than objects. The levels of the JDL model correspond nicely to the functions of our multiagent health monitoring framework. At Level 0, we have feature extraction, at Level 1, local classification and location, at Level 2, more global classification and location (involving hierarchies of monitor agents), and, at Level 3, prognosis. With a multiagent system, Level 4 occurs spontaneously and continuously through negotiation.

The JDL model embodies some strong realist assumptions, especially in distinguishing between Level 0, where information is handled without regard to its referents, and Level 1, where events and objects in the real world are located and classified. Despite this realism, much of the data-fusion literature pays little

regard to real-world referents. As an antidote to this myopia, we suggest a realist picture of perception, such as Gibson's ecological theory of visual perception [21]. What we directly see according to Gibson is the layout of objects by virtue of the stimulus information in the structure of ambient light. This realist position relates to JDL Level 2. A central concept for Gibson is that of the *affordances* of the environment: what behavioral possibilities it furnishes for an animal. In perceiving affordances, Gibson maintains, we perceive values: some offerings are beneficial, some injurious. This realist interpretation of Level 3 is particularly relevant to health monitoring.

We have agents distributed on a platform, perceiving features of events taking place in it and collaborating through speech acts (messages with performatives) to arrive at a diagnosis. To disentangle the threads here, consider Dennett's account of various strategies used to predict the behavior of systems [22]. The physical strategy views a system in terms of the physical sciences. If a system is too complex for the physical strategy, we may use the design strategy: predict that the system will behave as it is designed to behave. Finally, when even the design stance is impractical, there is still the intentional stance, treating the system as a rational agent: determine its beliefs and desires, then predict behaviors that will further its goals. Although the intentional stance (applied to computational entities) has been a cornerstone of agent research, our focus is different from Dennett's. While Dennett is concerned with the status of beliefs, we are interested in agent coordination hence in (socially situated) communication acts. (Still, such acts express beliefs and desires.) All three strategies dovetail together in our framework. The physical strategy is appropriate for addressing physical events and materials, the design strategy applies because a design is instantiated in the structure and known by the agents (in fact, it provides affordances), and the intentional strategy comes in because we are concerned with communication acts that express beliefs and desires.

Focusing on the social aspect, the key issue is how the agents coordinate to reach a diagnosis given the events they perceive together and the information they share about the design of the platform. Now, common knowledge can be shown to be a necessary condition for coordination [23]. Where ϕ is a proposition and G is a group, it is common knowledge in G that ϕ if everyone in G knows that ϕ, everyone in G knows that everyone in G knows that ϕ, and so on for arbitrarily deep nestings of the operator "every one in G knows that." Clark and Carlson [24] have presented heuristics for inferring common knowledge in terms of situations shared by agents A and B. The physical co-presence heuristic comes into play when an object is located between A and B and both see (or otherwise perceive) the object and each other simultaneously. With the linguistic co-presence heuristic, there is a triple co-presence of A, B, and the linguistic positing of the object of common knowledge. Finally, by the community membership heuristic, if A finds that B is in the same community as A, then A may assume that B has that community's common knowledge. All three of these heuristics apply to our agents. Community membership is designed into the system, and the agent communication language is designed to achieve linguistic

co-presence. Physical co-presence applies because several agents can perceive an event and, by design or arrangement, they have information about their neighbors' locations. Realism regarding perception makes it appropriate to apply this heuristic. Model checking fits nicely into this picture: the synchronization that is central to the abstract software design being verified is an abstraction from the system that is the object of the common knowledge that makes the coordination among the agents possible.

7 Conclusion

We formulated a multiagent structural health monitoring architecture that includes system agents, data-management agents, and alarm-raising supervisor agents bound to groups of sensors. There are also monitor agents, which coordinate the diagnosis task and configure themselves hierarchically. Specialty agents provide specific services by incorporating specialized software. The contract net protocol is used both to assign responsibilities and to assign problems on the fly. Since this highly concurrent architecture is too complex and uncontrollable for thorough testing, we used model checking to verify abstract designs of critical aspects against LTL specifications. Finally, a prototype to classify events related to fatigue crack growth was built. We related the tasks in our framework to data-fusion levels and Gibson's realist position on direct perception of objects and affordances. We showed how a system consisting of a multiagent system and the monitored platform exhibits behavior at several different levels, from the physics of acoustic emissions to inter-agent conversations expressing desires and beliefs. Communication, perception of public events, and system design conspire to provide the common knowledge in the agent community needed to coordinate diagnostic tasks. We intend to work on a multi-level interpretation with systematic integration of social and ecological concepts with each other and with concepts at the physical and design levels. This will reduce the conceptual distance between the physical events and awareness of the impact of the current situation on the success of the mission. Future work will also involve more extensive prototypes, and more aspects of the architecture will be model checked.

References

1. Garga, A., Campbell, R., Byington, C., Kasmala, G., Lang, D., Lebold, M., Banks, J.: Diagnostic reasoning agents development for HUMS systems. In: American Helicopter Society 57th Annual Forum, Washington, D.C. (2001) 1268–1275
2. Garga, A., McClintic, K., Campbell, R., Yang, C., Lebold, M.: Hybrid reasoning for prognostic learning in CBM systems. In: IEEE Aerospace Conference, Big Sky, MT (2001) 2957–69
3. Roemer, M., Vachtsenavos, G., Byington, C., Kacprzynski, G.: Two-day PHM/ CBM design course (2004) Orlando, FL.
4. Faas, P., Schroeder, J., Smith, G.: Vehicle health management research for legacy and future operational environments. Aerospace and Electronic Systems Magazine IEEE **17**(4) (2002) 10–16

5. Esterline, A., Gandluri, B., Sundaresan, M., Sankar, J.: Verified models of multiagent systems for vehicle health management. In: 12th SPIE Annual International Symposium on Smart Structures and Materials, San Diego, CA (2005)
6. Wooldridge, M.: Intelligent Agents. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT press, Cambridge, MA (2000) 27–78
7. Huhns, M., Stephens, L.: Multiagent Systems and Societies of Agents. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (2000) 79–120
8. Mabry, S., Schneringer, T., Etters, T., Edwards, N.: Intelligent agents for patient monitoring and diagnostics. In: Proceedings of 2003 ACM Symposium on Applied Computing, Melbourne, FL (2003) 257–262
9. Lam, S., Cao, J., Fan, H.: Development of an intelligent agent for airport gate assignment. Journal of Airport Transportation **7**(2) (2002) 103–114
10. Logan, K.: Prognostic software agents for machinery health monitoring. In: IEEE Aerospace Conference. (2003) 3213–3225
11. Franke, J., Satterfield, B., Czajkowski, M., Jameson, S.: Self-awareness for vehicle safety and mission success. Technical report, Lockheed Martin Advanced Technology Laboratories, Camden, NJ (2002)
12. Smith, R.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers **C-29(12)** (1980) 1104–1113
13. Shen, W., Norrie, D.: An agent-based approach for dynamic manufacturing scheduling. In Nwana, H., Ndumu, D., eds.: 3rd Int. Conf. on the Practical Applications of Agents and Multi-Agent Systems, London, UK (1998) 533–548
14. FIPA: Welcome to FIPA! http://www.fipa.org/ (2005)
15. TILab: Java agent development environment. http://jade.tilab.com/ (2005)
16. Mller, S.: JMatLink: Matlab Java classes. http://www.held-mueller.de/JMatLink (2005)
17. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, Boston,MA (2004)
18. Sundaresan, M., Nkrumah, F., Grandhi, G., Derriso, M.: Identification of failure modes in composite materials using a continuous ae sensor system. In: IMECE 2004: ASME Int. Mechanical Engineering Congress, Anaheim, CA (2004)
19. Steinberg, A., Bowman, C.: Revisions to the JDL Data Fusion Model. In: Handbook of Multisensor Data Fusion. CRC Press, Washington, D.C. (2001) 2–1–18
20. Elmenreich, W.: Sensor Fusion in Time-Triggered Systems. PhD thesis, Technical University of Vienna, Vienna, Austria (2002)
21. Gibson, J.: The Ecological Approach to Visual Perception. Lawrence Erlbaum Associates, Hillsdale, NJ (1986)
22. Dennet, D.: True Believers: The Intentional Strategy and Why It Works. In: The Intentional Stance. The MIT Press, Cambridge, MA (1987) 13–35
23. R. Fagin, J. Y. Halpern, Y.M., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge, MA (2003)
24. Clark, H., Carlson, T.: Speech Acts and Hearers' Beliefs. In: Mutual Knowledge. Academic Press, London (1982) 1–36

Towards a Model Level Debugger for the Cougaar Model Driven Architecture System

Boby George, Shawn A Bohner, and Nannan He

Department of Computer Science,
Virginia Tech Polytechnic Institute and State University,
Blacksburg Virignia, 24061
{boby,sbohner,nhe}@vt.edu

Abstract. Model-driven development generates software artifacts from abstract model representations through a series of specification elaboration and refinement activities. Yet, the resulting systems must still be debugged at the source code level; leaving it to the developer to associate the symptoms found during debugging with the root causes in the models. Abstractions in the model result in difficulties correlating between models (where changes are made) and the artifacts (where effects of the changes occur). This paper examines the issues involved in debugging systems produced using a model-driven approach and proposes an architecture for a model level debugger based on our work on the Cougaar Model Driven Architecture (CMDA). We present an approach that mimics current debuggers by storing the relevant mapping and transform information in the model repository during generation and then using it to construct the debugging structures needed for examining the program during execution from the model perspective.

1 Introduction

As we continue to evolve software development to generate software from more abstract representations (i.e., models), there is an increasingly significant disconnect between debuggers and the models that developers use to generate the software. While some commercial and experimental environments attempt to make this connection, the absence of debugging capabilities at the model level is hindering the progress in generating software systems. With highly complex and even radical applications of agents in today's emerging systems, this gap between debuggers and models used to produce these systems must be addressed. The debugger-to-model chasm is becoming painfully clear in Model-Driven Architecture (MDA – also known as Model-Driven Development). Moving from abstract Computation Independent Models (CIMs) to Platform Independent Models (PIMs), and ultimately to Platform Specific Models (PSMs), is a long way from the output of a typical debugger. Yet with MDA, the changes to "misbehaving software" will not be performed in the lowest level source code, but rather in the models. Hence there is a need to map the outputs of the debugger to the source code and ultimately to the models where the changes would be made. Without connecting debugging with the models, software engineers are left to manually map the symptoms shown in the code to the root causes in the models.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 86 – 97, 2006.

In a typical software debugging activity, the software engineer prepares the environment to run a program in a debug mode and interacts with source code represented in the debugger. The debugger maps the language (typically 3rd generation language – 3GL), to the breakpoints, registers, and the like that are monitored in the debugging environment. However, when the "programming" is at the more abstract model level, the software engineer is left to manually map the information from the debugger to the relevant models.

Over the past two years we have applied the MDA approach to improving the productivity of Cognitive Agent Architecture (Cougaar) development teams. While there are substantial improvements using the Cougaar Model Driven Architecture (CMDA) approach [1], there are disconnects between the resulting code execution and the relevant models used to generate these systems. We believe that the ability to effectively debug agent based systems, particularly the more sophisticated ones that are emerging today, will require support for resolving problems (detected in the operation) at the model level itself. For example, an agent is produced through a series of models and the behavior in the resulting software is incorrect. So, the developer runs a debugger to pinpoint where the problem is first manifest. While the debugger helps find where in the code the problem shows up, this is sometimes several models removed from where the syntactic or logical error was induced. Hence, the developer goes through a series of model explorations to determine at what model level the change can be successfully made. This can be improved by taking advantage of the roundtrip engineering support espoused in MDA [2]. The mappings and transformations provide dependency relationships that connect the identified debug point and its relationship to each model that was used to produce that piece of code.

The objective of this research is to provide an extension to the debugging facility where the model pieces can be depicted along with the debugged code. We believe that this would greatly enhance the developers' ability to diagnose certain types of problems that are often troublesome for agent developers. Therefore, in this research we examine a model-level debugger for CMDA based systems. We are hopeful the lessons we learn in this research generalize to the MDA and agent architecture communities at large. Anticipating the increasingly complex tasks that radical agent architectures will demand, we believe that debugging will concomitantly become important to identify causes of identified problems. While source code debuggers help with identifying where in the code that things go wrong, we do not have good mechanisms to connect these observations with the models in CMDA where the changes to the system are made.

2 Model Driven Architecture

Model Driven Development (MDD) provides the ability to create models (at various levels of abstraction), systematically refine and elaborate them, and provide automatic (or semi-automatic) translation to one or more execution platforms. The Object Management Group (OMG) has specialized this concept a bit and led the way with MDA advocating Unified Modeling Language (UML) as the modeling technology at the various levels.

In some sense, MDA is a natural progression from previous advances in computer science. Using models in the development of a system has been practiced for decades, and even for centuries in other engineering disciplines (e.g., Building Architecture). Perhaps the most telling transition in mindset is how modeling in MDA takes a model (typically an abstraction of a reality) and creates an executable form through a series of predictable transformations. Since the computer uses a conceptual medium developed by a software engineer (i.e., a model or series of models), transforms now make abstractions of the real world accessible and even executable on a computer. In this sense, models are no longer simply an aid in understanding — the model now becomes something much more concrete.

Like other engineering disciplines, software architecture helps us deal with the inherent complexities of building today's software systems. Systematically, separating concerns, formalizing the interfaces through standards and the like, provides better leverage for developing and evolving the software we employ. Software architecture – the structure or structures of the system, which encompass software components, their externally visible properties, and the relationships among them – addresses the aforementioned growing complexity by providing structure for thinking about and communicating key relationships between components, whether they are Commercial-Off-The-Shelf (COTS) software, middleware, or custom developed.

MDA endeavors to achieve high portability, interoperability, and reusability through architectural separation of concerns. In some respects, MDA is an advanced perspective on well-known essential systems development concepts practiced over the years (albeit frequently practiced poorly). MDA hinges on the long-established concept of separating the operational specification of a system from the details of how that system implements those capabilities on its respective platform(s). That is, separate the logical operational models (external view) from the physical design for the platform implementation.

Starting with an often abstract CIM such as a business process workflow or functional description, the PIM is derived through elaborations and mappings between the original concepts and the PIM renderings. Once the PIM is sufficiently refined and stable, the PSMs are derived through a further series of elaborations and refinements. The PSMs can then be transformed into a completed operational system.

The CIM layer is where vernacular specific to the problem domain is defined, where constraints are placed on the solution, and where specific requirements reside. Artifacts in the CIM layer focus largely on the system requirements and their environment to provide appropriate vocabulary and context (e.g., domain models, use case models, conceptual classes). The CIM layer contains no processing or implementation details.

The PIM provides the architecture, the logical design plan, but not the execution of the plan in a tangible form. Beyond high level services, the problem domain itself must be modeled from a processing perspective. The PIM is where the logical components of the system, their behaviors, and interactions are modeled. PIM artifacts focus on modeling what the system should do from an external or logical perspective. Structural and semantic information on the types of components and their interactions (e.g., design classes, interaction and state diagrams) are rendered in UML, the defacto modeling language for MDA.

Mapping from the PIM to the PSM, is a critical element of the MDA approach. The mappings from platform independent representations to those that implement the features or functions directly in the platform specific technologies are the delineation point where there is considerable leverage in MDA. This mapping allows an orderly transition from one platform to another. But the utility does not stop there. Like the PIM, there is the opportunity to have layers within the PSM to produce intermediate-transformations on the way to the executable system. These models can range from detailed behavior models to physical source code used in the construction of the system.

Direct PIM to PSM mappings are only possible in relatively simple situations today. Today's modeling languages are not sufficient to express all possible processing mechanisms. While UML 2.0 in MDA is a reasonable attempt to address this limitation, there are still significant hurdles before it is applied in the large. However, it is valuable for our perspective of a model level debugger. In the following section, we examine debugging as it applied to model level debugging.

3 Debugging and Model Debugging

Debugging software is a relatively mature activity with most commercial compilers providing some level of debugging as a standard part of their product [3, 4]. Indeed, this aspect of a compiler becomes an important factor in employing a language in the development of a system. Moreover, most competent programmers and software engineers are facile at using debugging software to identify root causes to problems [3]. These debuggers provide functions to track the values of variables, step through a program (single-stepping), stop at some kind of event by means of breakpoint, and to modify the state of the program (update variables and the like) while it is running.

As important as debuggers are, they are somewhat limited. Debuggers often change the internal timing of a software program; making it difficult to track down runtime problems in complex multi-threaded or distributed systems. Most research today on debuggers centers on this aspect. With the advancements in reusable components and highly distributed systems using the web, there has been a move towards pervasive debugging [6] – the ability to deal with both horizontal and vertical debugging. While this is addressing an important trend, we find that another important trend is the movement towards "programming-in-the-more-abstract." That is, the level of programming is moving up to more abstract models and the debugger must address this concern.

The need for model level debugging is important in MDA as correlating the models with the generated artifacts can be a tedious task. The task is difficult due to the wide semantic gap between the high-level models and the generated source code. The problem is aggravated by the automatic code generators used in MDA As a result of internal transformations and optimizations, the generators often produce code that, is not easily traced to the original model. Consequently, model-level observability – ability to trace errors in the generated code to the original models, has become a key issue for MDA [7]. The practical difficulties encountered in diagnosing problems arising of uncorrelated code and models can nullify the advantages of MDA. Programmers faced with the task of modifying code and reflecting those changes into the

model are more than likely to be discouraged from relying on the models. Hence for MDA to be successfully adopted, model level error reporting and debugging facilities are needed [5, 7].

There is some related work in model debuggers [5] where models of software systems are examined and debugged. This assumes an executable rendering of the model, but is frequently a simulation rather than the actual operational software source. Some researchers have used this approach with UML renderings of systems in real-time applications where the temporal aspects of software can be examined [5]. While these serve to help the MDA community, we are endeavoring to take this a step further and connect the models through MDA mappings and transformations to the actual source code being executed.

4 Cougaar Model Driven Architecture

The Cougaar is an open source, distributed agent architecture resulting from over eight years of research and development, and over a $150 million investment by the Defense Advanced Research Projects Agency (DARPA) under the Advanced Logistics Program (ALP) and the Ultra*Log program [8]. The CMDA system simplifies Cougaar-based application development by generating key software artifacts from the models. CMDA has two important abstraction layers namely Generic Domain Application Model (GDAM) and Generic Cougaar Application Model (GCAM). The GDAM reflects the PIM and encompasses the representation of generic agent and domain specific components found in the domain workflow. The GCAM layer, upon which the GDAM layer is built, reflects the PSM or Cougaar architecture, its specifications and environment.

Fig. 1 illustrates the basic CMDA approach. At the CIM level, the user specifies the workflow of the intended Cougaar system using XML Process Definition Language (XPDL). Then the user maps the workflow of the intended system into its PIM and PSM using GDAM and GCAM components respectively. The GDAM and GCAM components are stored in a repository. An assembly approach is used, whereby the developers assembles the system and implementation models of the intended Cougaar based system by choosing, configuring and connecting various predefined GDAM and GCAM components. Once completed, the models are then fed into a transformer which then parses through this assembled set of models to produce the actual software artifacts such as requirements, design, code and test cases. The generation of software artifacts is controlled by predefined mapping rules and template structures.

It is important to note that a typical Cougaar application is not completely generated from models and specifications. We estimate that 70% will be from GDAM, 20% from GCAM, 8% from Cougaar directly, and 2% using Java directly. Where there does not exist a component at the given level, that component is constructed from components at the next lower level. This recursive approach is depicted in CMDA meta-model.

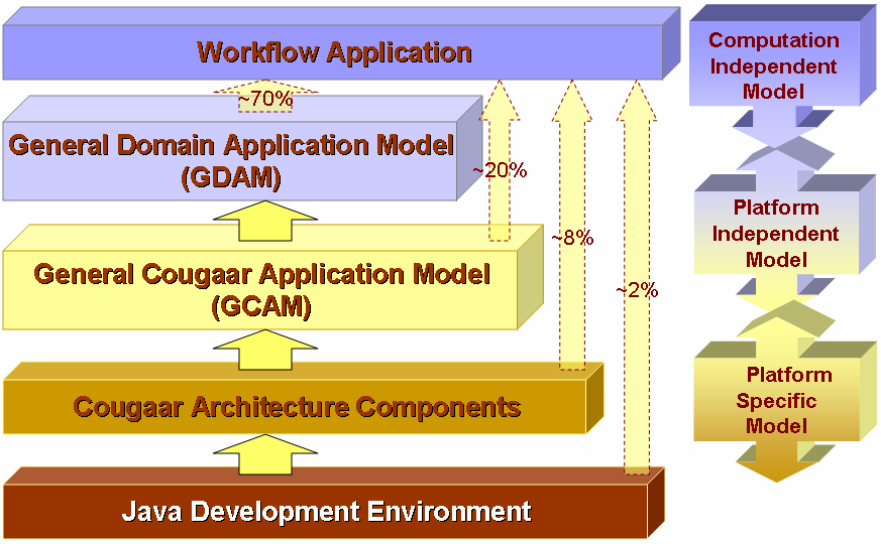


Fig. 1. The different abstraction layers present in the CMDA approach [1]

The meta-model defined using Eclipse Model Framework (EMF) is a critical aspect of the CMDA system. The meta-model, shown in Fig. 2, is the underlying representation for defining GDAM and GCAM model elements. At its core, the meta-model has an entity named component. It is configured for use by specifying a set of parameters. The leaf component (the component at the lowest level) names a mapping profile that links to the template used by the transformer for generating the required artifact. The instantiation is a reference to a component combined with values for its

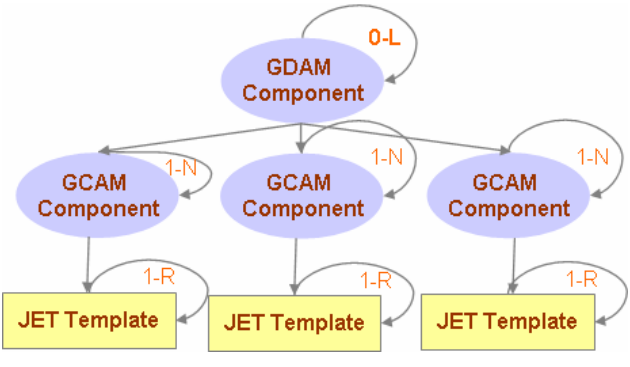


Fig. 2. Recursive meta-model forming the underlying representation

parameters. A parameter provides a declaration of the required input data. It allows hierarchical specification and allows optional, required, multiple, and singular values. Parameters have constraints, specified in Object Constraint Language (OCL), that

validate values assigned to them. A role is a parameter that specifies another instantiation to build connections between instantiations.

The same meta-model was used to define GDAM and GCAM to allow smooth translation between the two layers and to facilitate multiple sub-layers within the layers. The meta-model is recursive in nature and allows the users to specify the intended model as a hierarchy of components. Each component contains instances of other components, at same or lower layer. The leaf components are attached to the templates that contain information on how to process the parameters of the component and generate the required artifacts. For generating the code artifacts Java Emitter Templates (JET) were used.

4.1 Debugging in CMDA Context

The CMDA design approach lends itself to the debugging task through the mappings and transformation approach employed. The semantic gap between the models and generated code is less for CMDA than for typical MDA approaches, since the JET templates are linked to the leaf GCAM model elements. The mappings are derived from an intermediate file that contains component references and the relevant parameters. During the generation of a system, this information is recorded in the repository for future use in roundtrip engineering and in debugging.

There is still a need for model level debugger. For example, without the model debugger, once a fault is found in the source code, the developer has to correlate the fault to its possible location in the model level in his or her mind. It is important to recognize that the CMDA model debugger will enable developers to validate the system at a higher abstraction - the model level, just like the CMDA approach enabled developers to develop systems from the model level.

5 Design Approaches

The architecture for the CMDA model level debugger was decided after studying two prominent design approaches for model debugging [5, 9]. The first approach envisioned the ability to directly execute and debug the models right before the models are transformed into the required artifacts. In order to directly execute at model level, the models have to be represented using executable languages like executable UML [10]. In this approach, debugging becomes a part of the verification phase and is performed at certain level of segregation from the actual artifacts. Hence the developers will have to resort to normal debugging methods to triangulate the faults observed in the generated artifacts. The second approach is to execute the generated code and propagate the outputs back into the models. In this approach, the debugging is moved upwards into the higher abstract level – the model level. While the approach is harder to implement, as it requires roundtrip engineering, it eliminates the need to perform both model debugging and artifact debugging. The second approach is analogous to traditional debuggers where the machine code is executed and the results are traced back to the source code.

Nucleus Bridgepoint [5] uses the first approach of using executable model. However, we decided to forgo that approach based on concerns about accuracy

and necessity to perform transformation from EMF models to OMG-compliant eXecutable and Translatable UML (xtUML). The concern about the accuracy of transformation comes from the need to develop and maintain mapping criteria and transformation rules for each model elements. The problem of necessity occurs because of the need to develop an entire new set of models and make them fit into the existing CMDA architecture.

The second approach of executing the actual code and propagating the results back into the models calls for development of dependency and traceability techniques to keep the software artifacts and models consistent. Since most of the requisite information is already recorded (into models and templates) during the transformation of model into software artifacts, the approach can be implemented by developing a mechanism to trace the results to the correct model levels.

5.1 CMDA Debugger Design Approach

In order propagate the error up to the models, an analysis module named CMDA model level debugger, is hooked onto the conventional source code debugger. Fig. 3 delineates the design approach for the proposed CMDA model level debugger. The overall working of the approach is explained as follows: the CMDA system retrieves the models from the CMDA repository and generates the source code, which is then compiled into machine code. During compilation, the compiler inserts instrumentation code and also consults/updates the compiler symbol table. The machine code is then executed and upon triggering a breakpoint or fault, the debugger halts the execution of the application. The debugger records the execution state of the application and correlates it with the source code. Then the information is passed on to the model level debugger. Just as the source code debugger correlates between source code and machine code, the model level debugger correlates between source code and the different models.

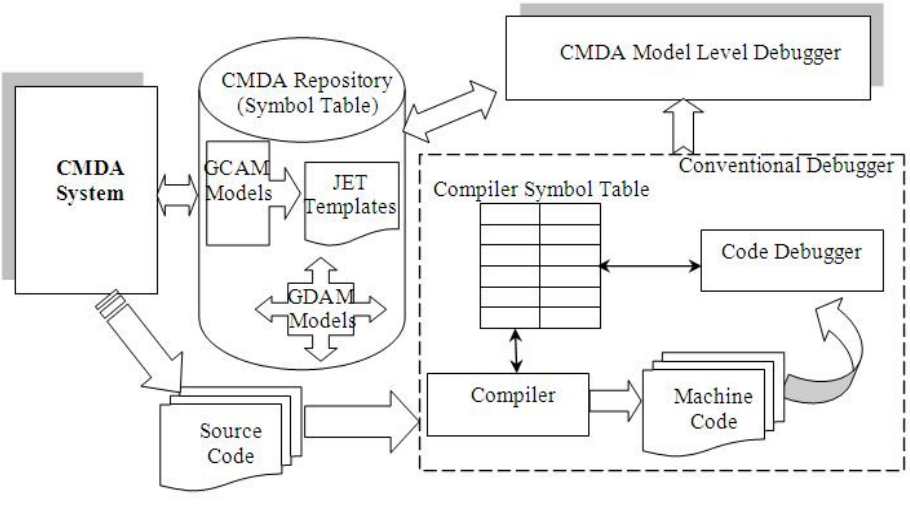


Fig. 3. CMDA model level debugger design approach

To correlate between the source code and model, the analyzer must be aware of all the dependencies between the source code and the models. These details are obtained from CMDA repository which acts as the symbol table for the model level debugger. The repository stores all the GDAM models, GCAM models, templates and their interlinking dependencies. The model debugger pulls in all the source code that was being executed and finds the interlinking model elements by searching the repository. Thus the model debugger superimposes and correlates the current state of the application with all the related models, providing the developer with a model perspective on the debugging.

6 CMDA Debugger Architecture

The CMDA system was implemented as a series of plugins for the Eclipse IDE. The Eclipse platform provides a very powerful framework that provides support for extending or adding new debuggers without too much effort [4]. The general architecture of the proposed CMDA debugger is shown in Fig. 4. The Eclipse SDK provides a framework (collectively known as the debug platform) for building and integrating debuggers. The debug platform defines the debug model, a set of Java interfaces modeling a group of artifacts (like threads, stack frames) and actions (like stepping, resuming) common to debuggers [4]. The execution interface extends the standard Java debugger and obtains the debugging context information, which is passed on to the debugger manager. The debugging manager has two important modules GDAM and GCAM. As their name refers, each of the modules retrieves the respective model elements from the CMDA repository and populates the models with the debugging context.

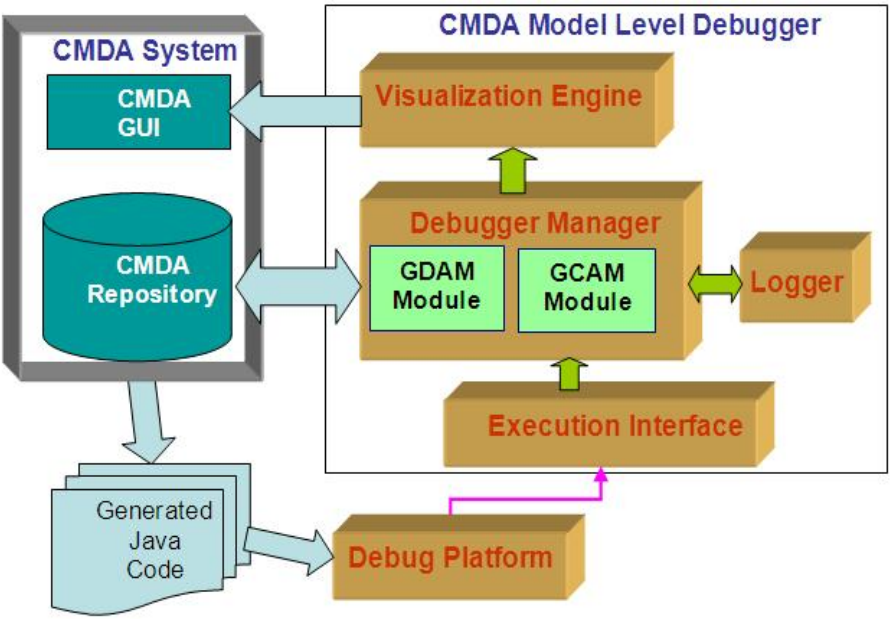


Fig. 4. General architecture of the proposed CMDA model level debugger

Since the leaf components of the models have links to templates, it is responsibility of the GCAM module to perform the matching between the source code and its corresponding templates. A series of techniques from as trivial as matching the name of source code file (each root template has information on how file name has to be generated) can be used for the same. Once the appropriate template and GCAM model is found, the module tries to populate the model with the debugging context. GDAM module performs the matching and population of the GDAM model. Matching of the GDAM module is trivial as the linkage is clearly defined. Upon reaching the topmost GDAM component or when an error in matching occurs, the modules output the model information into the visualization engine. The visualization engine modifies the appropriate user interface model (GEF is the underlying framework of CMDA GUI) to display the mapped models in the CMDA GUI. The logger is a standard logging module that records all the mapping activities performed by the debugger manager. The logger also stores configuration information that the debugger manager is interested in. Since false positives and mismatches are likely to occur, going through the log report can provide developers with useful information on how to enhance the model level debugger.

6.1 Limitations

As Cougaar is a multi-agent software application, the limitations of most conventional debugging tools for multi-agent software systems are applicable to the CMDA model level debugger. A key limitation of debuggers for multi-agent system is that they have huge overhead and output too much raw data, thereby lowering the comprehensibility and usability. Further, the dynamic nature of agents' compounds to the problem of elucidating the precise information required for triangulating where the defects might be [11]. Furthermore, some of the limitations present in source code debuggers are likely to be aggravated in CMDA debugger due to the abstractions involved. These include changes to the temporal nature of the software due to breakage in execution and changes to program logic due to the introduction of instrumentation code. There is a high probability for CMDA debugger to generate false positives while the models are populated with the debugging context. However, we believe the generation of false positives and other limitations of the debugger will get contracted as it evolves.

7 Conclusion

With the trend towards model-driven development, there is a concomitant need for de model-level bugging environments. While this research is in it's early stages, it shows promise with using the artifact generation information from the models to support the debugging environment. Analogous to the typical compiler, the CMDA debugger uses the elaboration and refinement information from the models to reference the code that is debugged. Code to model dependencies are represented statically.

There are some limitations to our approach. First, the correlation between the source code debugger and the model(s) is highly dependent on the dependency information collected during model elaboration and refinement. The debugging precision

is a function of how well the information is collected and conditioned for the debugging environment. This means that the mappings and transformations must be conducted with "debugging in mind." We will need to experiment with the prototype to evolve our representation and conditioning approaches. Also, the propagation can be compute intensive as there is a lot of information that must be analyzed and rendered. We will need to overcome this to provide near real-time stepping through the code and model. The user interface is aligned with debugging in the current architecture, but future versions of this capability must support things like stepping through the model (break points at model boundaries).

This research is the first step towards a full model level debugger for CMDA. Upon this we plan to build a pre-conditioner for debugging information to improve performance and accuracy. Further, we plan to build on the typical functions of traditional debuggers and experiment with features like animated tracing of the execution from the perspective of the models and model level breakpoints. With a model level debugger that provides features at the level of the models, model-driven development should be more effective.

Acknowledgements

We are indebted to several people for their help as we prepared this paper. We would like to thank Denis Gracanin, Lally Singh, and DongKwan Kim for their valuable time and thoughts regarding this project. We would also like to thank Cougaar Software, Inc. for their support in this research.

References

1. Bohner, S., George, B., Gracanin, D., Hinchey, M.,: Formalism Challenges of the Cougaar Model Driven Architecture, Proceedings of the Third Formal Approaches to Agent-Based Systems Conference (FAABS III), also in Lecture Notes in Comp. Science, Springer-Verlag GmbH, (2005).
2. OMG: Model Driven Architecture, URL: www.omg.org/mda. [Last access: 11/11/05]
3. Matloff, N., Guide to Faster, Less Frustrating Debugging, University of California at Davis, (2002) URL: http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html [Last accessed: 11/11/05]
4. Wright, D., Freeman-Benson, B., How to Write an Eclipse Debugger, Eclipse corner article (2004) URL: http://www.eclipse.org/articles/Article-Debugger/how-to.html [Last accessed: 11/11/05]
5. Nucleus Bridgepoint: The BridgePoint Model Debugger, Accelerated Technologies Inc., URL: http://www.nrt.se/nrt/uml/Produktinfo/BridgePointModelDebug.pdf [Last access: 11/11/05]
6. Ho, A., Hand, S., Harris, T., Pervasive Debugging, AADEBUG'05, September 2005.
7. Selic, B.,: The Pragmatics of Model-Driven Development IEEE Software 20, 5 (2003) 19-25
8. BBN Technologies, COUGAAR Architecture Document for Cougaar Version 11.4, (2004) URL: http://cougaar.org/docman/view.php/17/170/CAD_11_4.pdf [Last accessed: 11/11/05]

9. Jacobs, T., Musial, B.: Interactive Visual Debugging with UML, Proc. of the 2003 ACM symposium on Software visualization, (2003), 115-122
10. Raistrick, C.,: Model Driven Architecture with Executable UML, Cambridge University Press, (2004).
11. Poutakidis, D., Padgham, L., Winikoff, M.,: Debugging Multi-Agent Systems Using Design Artifacts: The Case of Interaction Protocols, Proc. of the 1st Int. Joint Conf. on Autonomous Agents & Multi-agent systems, (2002), 960-967.

Can Agent Oriented Software Engineering Be Used to Build MASs Product Lines?

Joaquín Peña

Dpto. de Lenguajes y Sistemas Informáticos
Avda. de la Reina Mercedes, s/n. Sevilla 41.012, Spain
joaquinp@us.es
www.tdg-seville.info

Abstract. On the one hand, the Software Product Lines (SPL) field is devoted to build a core architecture for a family of products from which concrete products can be derived rapidly by means of reuse. On the other hand, Agent-Oriented Software Engineering (AOSE) is a software engineering paradigms dedicated to build software applications composed of organizations of agents. Bringing AOSE to the industrial world may prettily benefit from SPL advantages. Using SPL philosophy, a company will be able to define a core MAS from which concrete products will be derived for each customer. This can reduce time-to-market, costs, etcetera. In this paper, we expose the similarities between AOSE and SPL concluding the viability of future research in Multi-Agent Systems Product Lines (MAS-PL).

1 Introduction

Agent-Oriented Software Engineering (AOSE) is a new software engineering paradigm that promises to enable the development of more complex systems than with current Object-Oriented approaches using agents and organizations of agents as the main abstractions [14]. In this field, agents are used as the main implementation artifact and organizations as the main artifact in modelling the system.

A software agent is a piece of software which exhibits the characteristics firstly described by Wooldridge in Ref. [26], namely: autonomy, reactivity, pro-activity and social ability. Autonomy means that an agent operates without the direct intervention of other agents or humans and has control over its actions and its internal state. Reactivity means that an agent perceives its environment and responds in a timely fashion to changes that occur in it. Pro-activity means that an agent does not simply react to changes in the environment, but exhibits goal-directed behaviour and takes the initiative when it considers it appropriate. Social ability means that an agent interacts with other agents (if it is needed) to complete its tasks and helps or contends with others to achieve its goals. Many researchers are in agreement with the vision of software agents as a characterization; and they, depending on their community, attach new attributes to software agents as mobility in distributed systems or adaptability in machine learning.

Since agents are limited to a specific environment and have limited abilities, complex problems are usually solved by a group of agents [4,5,24], that is to say, by a Multi-agent System (MAS hereafter).

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 98–108, 2006.

A Software Product Line (SPL) is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way, according to the definition used by the Software Engineering Institute (SEI) [6]. This engineering paradigm is focused in bringing reuse until its final outcomes. It is devoted to create software products, which are said to belong to a family of products, which present similar features, from a set of reusable software artifacts and a core architecture.

Both fields present many similarities. In this paper, we expose these similarities and we argue for the viability of integrating both approaches to enable the development of MAS Product Lines.

This paper is structured as follows: Section 2 shows the case study we use; Section 3 shows the features of AOSE that are important for our purpose; Section 4 shows the features of SPLs that are important in the AOSE field; Section 5 shows the main similarities between both and the deficiencies of AOSE to enable MAS-PLs; and finally, Section 6 shows our main conclusions.

2 A Case Study

The case study we have chosen has been proposed in the Modelling TC of FIPA with the purpose of evaluating how AOSE methodologies model the interaction aspect.

This case study is used along this document. Following, we present the original version of the problem[1]. The case study represents the UN Security Council's Procedure to Issue Resolutions. We use some parts of this example to show the similarities between both fields.

In addition, notice that, although it does not represent a product line, we will take some of the requirements as optional to obtain a product line where each product presents a subset of the requirements presented following.

Assumptions

(1) The following procedure is defined for a case study of agent-oriented modelling. It is inspired from the procedure of UN Security Council to pass a resolution. However, it does NOT necessary represents the reality.

(2) There are also some unspecified issues and ambiguity in the specification that models in different notations and languages may resolve by themselves in the process of modelling.

Description

The UN Security Council (UN-SC) consists of a number of members, where some of them are permanent members. Members become the Chair of the Security Council in turn monthly.

To pass a UN-SC resolution, the following procedure would be followed:

1. At least one member of UN-SC submits a proposal to the current *Chair*;
2. The *Chair* distributes the proposal to all members of UN-SC and set a date for a vote on the proposal.

[1] http://www.auml.org/auml/documents/UN-Case-Study-030322.doc

3. At a given date that the Chair set, a vote from the members is made;
4. Each member of the security council can vote either FOR or AGAINST or SUSTAIN;
5. The proposal becomes a UN-SC resolution, if the majority of the members voted FOR, and no permanent member voted AGAINST.
6. The members vote one at a time.
7. The Chair calls the order to vote, and it is always the last one to vote.
8. The vote is open (in other words, when one votes, all the other members know the vote)
9. The proposing member(s) can withdraw the proposal before the vote starts and in that case no vote on the proposal will take place.
10. All representatives vote on the same day, one after another, so the chair cannot change within the vote call; but it is possible for the chair to change between a proposal is submitted until it goes into vote, in this case the earlier chair has to forward the proposal to the new one.
11. A vote is always finished in one day and no chair change happens on that day. The chair sets the date of the vote.

3 Agent Oriented Software Engineering (AOSE)

In addition to the commonly used division in requirements, analysis, design and implementation, the software process of AOSE methodologies can be also divided into two phases, each of them used to describe the organization of the system from a different point of view [27]. In this section, we first show each kind of organization, to later discuss on the software process and models used in AOSE.

An organization represents a group of agents formed in the system in order to obtain benefits from one another in a collaborative or competitive manner [15,20,26]. Therefore, a multi-agent organization emerges when there exists some kind of interaction between its participants, either through direct communication or through the environment. As recognized in the field of economics [19], and other authors have recognized in the agent field, e.g. [3,11,27], an organization can be observed from two different points of views:

The interaction point of view: It describes the organization from the set of interactions between the roles played by agents in the system. The interaction view corresponds to the functional point of view in the field of economics.

The structural point of view: It describes the agents of the system and their distribution over sub-organizations, groups, and teams. In this view, agents are also presented in hierarchical structures showing the social architecture of the system.

In agency, the former is called *Acquaintance Organization*, and the latter is called *Structural Organization* [27]. Both views are intimately related, but they show the organization from radically different points of view.

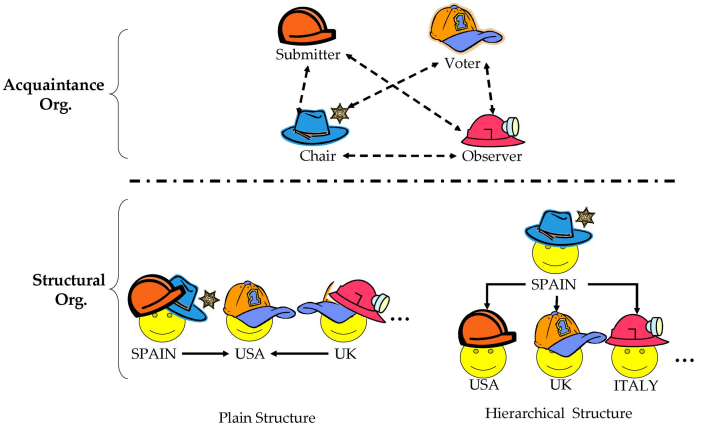


Fig. 1. Acquaintance *vs.* structural organization

Since any structural organization must include interactions between its agents in order to function, it is safe to say that the acquaintance organization is always contained in the structural organization. Therefore, if we determine first the acquaintance organization, and we define the constraints required for the structural organization, a natural map is formed between the acquaintance organization and the correspondent structural organization. This is the process of assigning roles to agents [27]. Thus, we can conclude that any acquaintance organization can be modelled orthogonally to its structural organization [18].

In Figure 1, we present a conceptual model of the case study presented in Section 2. Here we are representing only the procedure required to send, vote, withdraw and accept or reject resolutions not representing the possibility of changing the Chair, that is to say, an acquaintance sub-organization of our problem. Notice that for representing an acquaintance sub-organization we should have represented also the dynamic part of the system, that is to say, the behaviour of the roles in the sub-organization. However, we have not done that in order to simplify.

In the acquaintance organization, it implies a set of roles and interactions between them. In the structural organization, these roles can be mapped onto a certain group of agents to form several organizational structures. For example, as shown in Figure 1, we can map the acquaintance organization into a plain structure, a hierarchical structure, and so on. Thus, this exemplifies the fact that the structural organization of the UN Security Council is different and independent from its structural organization.

Most AOSE methodologies recognize this separation what derives in that some methodologies have proposed a software process where one phase appear for developing each kind of organization: *acquaintance analysis* and *structural analysis*[2]. This fact is ratified by the importance that the role concept, crucial for modelling the acquaintance organization, has reached in this field where most important methodologies

[2] Notice that these phases present a different name in each methodology, for example, in GAIA they are called analysis and architectural design respectively.

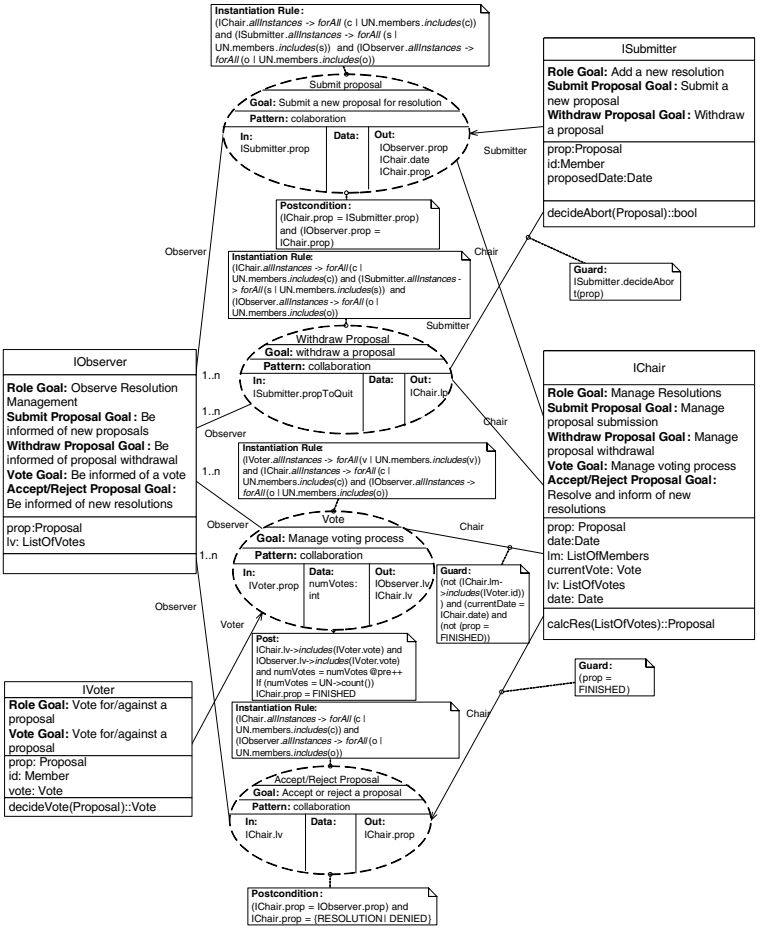


Fig. 2. Role Model for the Issue Resolution system goal

use it. Good examples are GAIA [27], INGENIAS [21], MASE [9], PASSI [2] and TROPOS [1].

For example, the MaCMAS methodology fragment [22], represents role models as it is shown in Figure 2. This model uses an extension of UML collaborations to represent the acquaintance organization. We can find two main types of elements in the model: (1) roles, represented as boxes; and (2) interactions, represented using ellipses. Roles show which are their general goals and their particular goals when participating in a certain interaction with other roles or with some part of the environment. Roles also represent the knowledge they manage, middle compartment, and the services they offer, bottom compartment. For example, the goal of the *Chair* role is "Manage resolutions", while its goal when participating in the *Withdraw proposal* interaction is to manage the process of withdrawing a proposal. In addition to

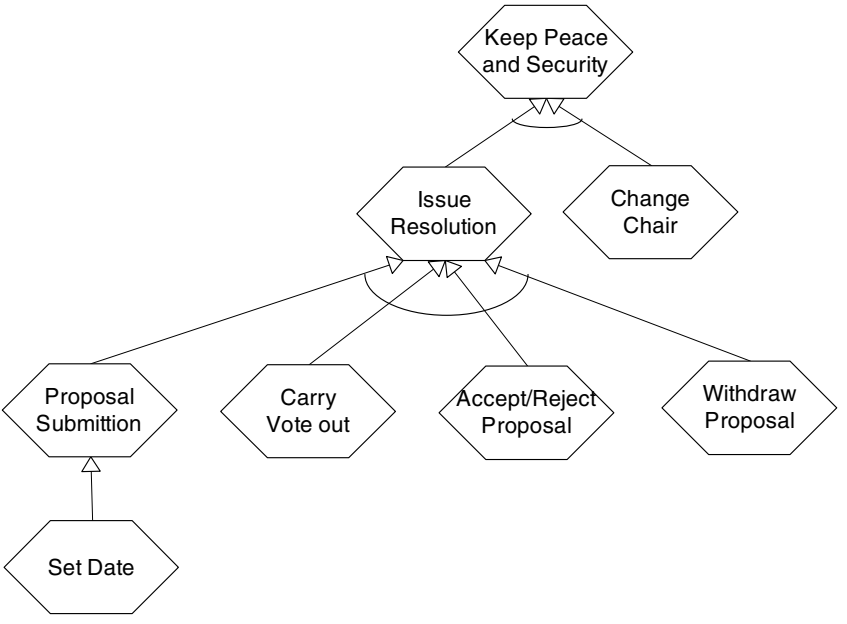


Fig. 3. Example of Goal Hierarchy using TROPOS

roles, interactions show us also some important information. They also must show the system-goal they achieve when executed, the kind of coordination is carried out when executed, the knowledge used as input to achieve the goal and the knowledge produced. For example, the goal of the interaction *Submit Proposal* is to "Submit a new proposal for a resolution". It is done by taking as input the knowledge of the *Submitter* about the proposal and producing as output the date of the proposal in the *Chair*, and the proposal in the *Chair* and in the *Observer*. Notice that as shown in Figure 1, these roles will be mapped onto agents in the phase devoted to build the structural organization.

In addition, many authors recommend the use of goal-oriented requirement approaches for this kind of systems. Goal-oriented requirements techniques analyse system goals producing a hierarchical diagram where each system goal is related with the system goals that decompose it [1,8,9,17,27]. Since agents are designed to fulfill goals, and organizations are designed to fulfill those goals that are sufficiently complex for requiring more than one agent to be satisfied, this information is also used to determine decomposition of the acquaintance organization into sub-organizations by some methodologies, for example, MASE, GAIA and MaCMAS.

In Figure 3, we show the hierarchical goal diagram of our case study using TROPOS. As can be observed, the general system goal of our case study is *Keep Peace and Security*. It shows also that to fulfill this goal the diagram we have to fulfill two finer grain system goals since they are related using an AND decomposition: the system goal *Issue Resolutions* and *Change Chair*. Notice that also OR relationships are allowed.

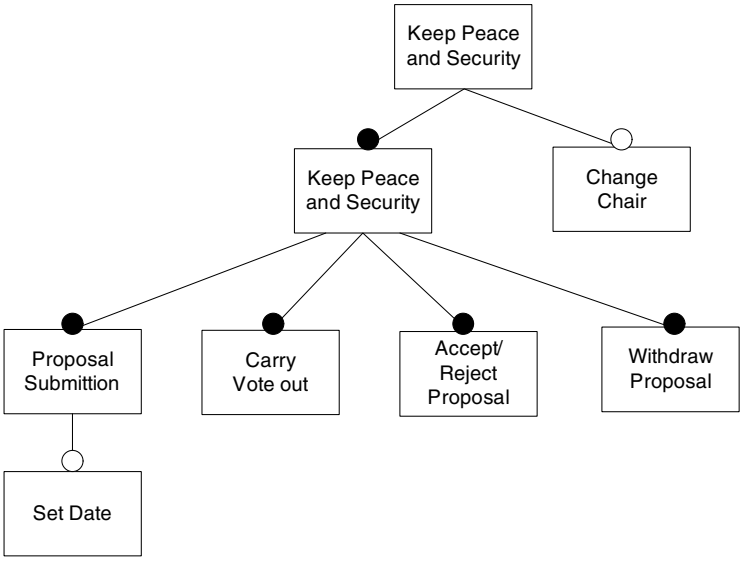


Fig. 4. Example of Feature Model

4 Software Product Lines (SPL)

The software process of SPLs approaches is usually divided in two main stages: *Domain Engineering* and *Application Engineering*. The former is in charge of providing the reusable core assets that are exploited during application engineering when assembling or customizing individual applications [12].

Domain engineering and application engineering phases are usually further divided into analysis, design, and implementation. From all these phases, we only detail those that can be applied in AOSE.

The analysis phase of domain engineering is applied to analyse the problem domain of a family of applications. It main purpose is identifying the common set of features that can be reused across all the applications that can be built under the domain at hand. Following, we show the most relevant stages for our purpose:

Domain scoping is devoted to describe the boundaries of the domain, which is crucial to determine the scope and needs of the product family. Another important stage is the *Commonality analysis*. It specifies the commonalities and points of variation in the domain that represents the first step towards a description of the family and the differences that each product may present.

Domain modelling is dedicated to produce documents where common and variable requirements are showed and the analysis models for these requirements. Thus, these documents specify which set of requirements are valid and may produce a valid application and which do not. One of the most accepted techniques to perform these documents are *feature models* [7]. A feature is a characteristic of the system that is observable by the end user [16].

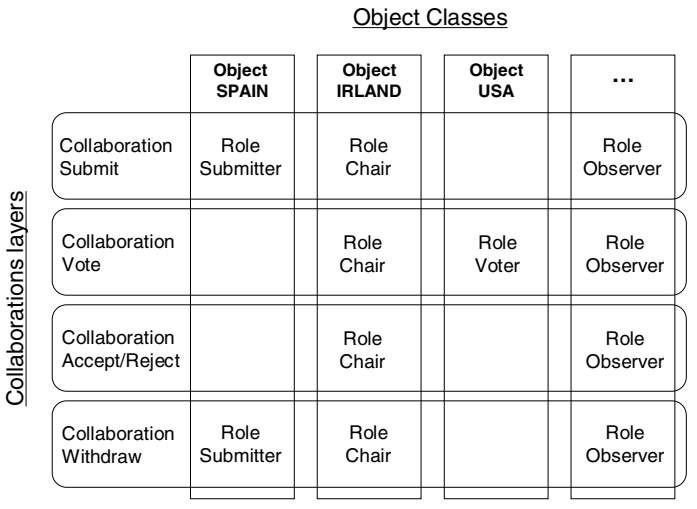


Fig. 5. Conceptual diagram of our case study using MIXIN layers

Feature models represent features hierarchically and its relationships. For example, in FODA, features in the model can be mandatory, optional, or alternative. In figure 4, we show an example of feature model representing our case study with some modifications. Filled circle indicate that the feature is mandatory, that is to say, if the parent feature is present in the product, it must be also present in the product. White circles indicates that feature is optional as it happens with the *Change Chair* feature. This means that we can have a product where the chair is changed and another product where there is no possibility of changing the chair. Although not showed in the example, we can also have filled arc indicating that for the final products we must have at least one of the features linked, and white arcs indicating that we can only have one of the features linked.

Finally, in the domain modelling stage, we can have also analysis models that show the interface(s) required in the system for including each feature. As we discuss below.

The design phase of domain engineering is devoted to model the core architecture. It consists in producing a core architecture by means of combinations of models that represent features that are common. Many approaches have appeared in the literature. From all of them, we are interested on those that are nearer to the AOSE field. In these approaches, role models are used to represent the interfaces and interactions that components of the system should provide to cover certain functionality (a feature in the features model). Later, role models are composed to produce the core architecture, the most representative are [13,25], but similar approaches has appeared in the OO field, for example [10,23].

Figure 5 shows a conceptual diagram of our case study extracted from [13]. There, authors show the mapping of domain models, collaborations, over the core architecture needed to build a certain product using OO programming. In this paper, authors use collaborations to represent features of the system. Collaborations are interaction-centered diagrams that show the interaction in a subsystem and the roles, sub-set of objects,

involved in these interactions. Later they propose to map them onto certain objects for building a concrete product, as shown in the figure using our case study.

Finally, notice that we do not discuss on the implementation phase of domain engineering since it does not correlate with agent implementations. We neither detail application engineering due to AOSE methodologies are not prepared to produce a family of MASs.

5 SPL/AOSE Correlation

The main points of correlation between both fields are three: (i) the correlation in the phases; (ii) the correlation in models used at requirements; (iii) and the approach followed to design a concrete MAS and to design a certain product. The main difference is that AOSE does not manage in the requirement phase the existence of several products. Following, we discuss first on similarities, to finish showing the main deficiencies of AOSE to enable a MAS-PL approach.

The first similarity is probably the most significant. This can be found in the phases of both approaches. On the one hand, the *analysis phase of domain engineering* in SPL is dedicated to define an architectural independent document describing the features of the system and the *acquaintance analysis* in AOSE is dedicated to define a set of acquaintance sub-organizations independently from the structural organization. Thus, as shown in previous sections, in both cases role models/collaborations are used to represent these independent models. However, SPL approaches finalize this phases implementing a core architecture, while AOSE approaches do not implement these models.

The second similarity consists in that requirement documents used for this phase in both fields present the same structure. Given that system goals in MASs are functional requirement observable by the end user, system-goals represent the same concept than features. Feature models, used in SPL, and goal-oriented requirement documents, used in AOSE, are both hierarchical where each feature/system-goal is decomposed in lower levels of the hierarchy indicating in both cases when they are mandatory, optional or alternative. Notice only some AOSE methodologies allow having mandatory, alternative or optional goals, but this feature is present in AOSE.

Third, the *design phase of domain engineering* in SPL is dedicated to produce the core architecture for the family of products and *structural analysis* is also devoted to define the structure of the organization of the MAS. In both field, role models/collaborations are used for defining architecture/structure independent models, and in both fields this process consists in composing models. If we concentrate on the SPL approaches that represent features using role models/collaborations, and given that in AOSE this phase consists in composing roles and the also use role models, both fields follows a quite similar procedure and models.

However, there exist some important differences between both fields mainly motivated by that AOSE is concentrated on developing a unique MAS and not a family of them. While in SPL the features that are composed to build the core architecture are those that are common for all products and composition is done guided by the commonality analysis, in AOSE roles are composed when they pursue similar goals or they provide a similar functionality.

As can be observed, current AOSE approaches align with SPL in their phases and models, but lacking from the artillery necessary to develop a general architecture instead of a unique product, namely, commonality analysis and domain scoping.

6 Conclusions

In this paper, we have shown the similarities and differences between SPL and AOSE. Given that AOSE methodologies follows a similar approach to SPL, where phases and models are quite similar, we can conclude that they may be extended to enable the development of MAS Product Lines (MAS-PL).

The main research lines that should be explored for enabling this kind of developments reside in: (i) adding commonality analysis and domain scoping to AOSE methodologies; and (ii); integrating the results of these activities to modify the process of assigning roles to agents and thus building a structural organization able to produce a family of MASs. In addition, the application engineering phase of SPL should also be included in AOSE methodologies to establish the procedures needed to derive a MAS from a MAS-PL.

References

1. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: an agent-oriented software development methodology. *Journal of Autonomous agents and Multiagent Systems*, 8(3), 2004.
2. P. Burrafato and M. Cossentino. Designing a multi-agent solution for a bookstore with the passi methodology. In *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002). CAiSE'02*, Toronto, Ontario, May 2002.
3. G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon, P. Kearney, J. Stark, and P. Massonet. Agent oriented analysis using MESSAGE/UML. In *Proceedings of Agent-Oriented Software Engineering (AOSE'01)*, pages 101–108, Montreal, 2001.
4. C. Castelfranchi. Founding agent's "autonomy" on dependence theory. In *14th European Conference on Artificial Intelligence*, pages 353–357. IOSPress, 2000.
5. C. Castelfranchi, M. Miceli, and A. Cesta. Dependence relations among autonomous agents. In In Y. Demazeau and E. Werner, editors, *Third European Workshop on Modeling Autonomous Agents in a Multi-Agent World. Decentralized AI 3*. Elsevier, 1992.
6. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.
7. K. Czarnecki and U Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison–Wesley, 2000.
8. A. Dardenne, A. van Lamsweerde, and S.Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.
9. S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *The International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001. World Scientific Publishing Company.
10. D.F. D'Souza and A.C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison–Wesley, Reading, Mass., 1999.

11. J. Ferber, O. Gutknecht, and F. Michel:. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *IV International Workshop on Agent-Oriented Software Engineering (AOSE'03)*, volume 2935 of *LNCS*, pages 214–230. Springer–Verlag, 2003.
12. M Harsu. A survey on domain engineering. Technical Report 31, Institute of Software Systems, Tampere University of Technology, December 2002.
13. A. Jansen, R. Smedinga, J. Gurp, and J. Bosch. First class feature abstractions for product derivation. *IEE Proceedings - Software*, 151(4):187–198, 2004.
14. N. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
15. N. R. Jennings. Agent-Oriented Software Engineering. In Francisco J. Garijo and Magnus Boman, editors, *Proceedings of MAAMAW-99*, volume 1647, pages 1–7. Springer-Verlag: Heidelberg, Germany, 30– 2 1999.
16. K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie-Mellon University, November 1990.
17. E. Kendall, U. Palanivelan, and S. Kalikivayi. Capturing and structuring goals: Analysis patterns. In *Proceedings of the* 3^{rd} *European Conference on Pattern Languages of Programming and Computing*, Germany, July 1998.
18. E. A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, April/June 2000.
19. H. Mintzberg. *The Structuring of Organizations*. Prentice-Hall, 1978.
20. H. V.n D. Parunak, S. Brueckner, M. Fleischer, and J. Odell. A design taxonomy of multi-agent interactions. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *IV International Workshop on Agent-Oriented Software Engineering (AOSE'03)*, volume 2935 of *LNCS*, pages 123–137. Springer–Verlag, 2003.
21. J. Pavón and J. Gómez-Sanz. Agent oriented software engineering with ingenias. In V. Marík, J. Müller, and M. Pechoucek, editors, *Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003, Prague, Czech Republic, June 16-18, 2003, Proceedings*, volume 2691 of *Lecture Notes in Computer Science*, pages 394–403. Springer, 2003.
22. J. Pena. *On Improving The Modelling Of Complex Acquaintance Organisations Of Agents. A Method Fragment For The Analysis Phase*. PhD thesis, University of Seville, 2005.
23. T. Reenskaug. *Working with Objects: The OOram Software Engineering Method*. Manning Publications, 1996.
24. J. S. Sichman, Y. Demazeau, R. Conte, and C. Castelfranchi. A social reasoning mechanism based on dependence networks. In Y. Demazeau and E. Werner, editors, *11th European Conference on Artificial Intelligence*, pages 416–420. John Wiley and Sons, 1994.
25. Y. Smaragdakis and D. Batory. Mixin layers: an object–oriented implementation technique for refinements and collaboration-based designs. *ACM Trans. Softw. Eng. Methodol.*, 11(2):215–255, 2002.
26. M. J. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, June 1995.
27. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, to be published 2003/2004.

Towards Dynamic Electronic Institutions: From Agent Coalitions to Agent Institutions

Eduard Muntaner-Perich and Josep Lluís de la Rosa

Agents Research Lab, University of Girona
Campus Montilivi (PIV), Girona, Catalonia, Spain
{emuntane,peplluis}@eia.udg.es

Abstract. In this paper, which is exploratory in nature, we introduce the concept of *dynamic electronic institutions*, which from our point of view arise from the convergence of two research areas: electronic institutions and coalition formation. We define dynamic electronic institutions as emergent associations (coalitions) of intelligent, autonomous and heterogeneous agents, which play different roles, and which are able to adopt a set of norms in order to interact with each other, with the aim of satisfying individual goals and/or common goals. These formations are dynamic in the sense that they can be automatically formed, reformed and dissolved, in order to constitute temporary electronic institutions on the fly. This type of institution should be able to adapt its norms and objectives dynamically in relation to its present members (agents). In this paper we introduce some preliminary ideas on how these systems could be developed, and present our first exploratory work. We argue that dynamic electronic institutions are potentially important in open-agent system applications.

1 Introduction

It is a fact that real-world applications are becoming increasingly complex, mainly as a result of the ever more significant role of Internet in our lives and the emerging model of electronic business. There are many application domains and scenarios in which autonomous, heterogeneous agents (software applications, enterprises, nodes, groups of people, etc) have to collaborate and engage themselves in temporary alliances that must change dynamically according to their environment and their current situation (markets, topologies, etc.).

Open multi-agent systems can be a good solution to these problems, and coalition formation mechanisms facilitate the appearance of these agent associations, but the emergent behaviour of the global system can become chaotic and unexpected. In critical applications this can be a significant problem, and it is evident that regulatory measures must be introduced to determine what things agents can and cannot do.

Electronic institutions could be an effective solution. They incorporate social and organisational abstractions into multi-agent systems; in fact they incorporate the rules of the game. But they have some limitations: they are based on medium to long-term associations between agents and require a design phase performed by humans; there are no mechanisms for reconfiguration and dissolution processes. A possible solution could be to use virtual organisations (which allow temporary collaboration between

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 109 – 121, 2006.

independent, heterogeneous enterprises), but these systems have domain-dependent infrastructures and architectures for the B2B electronic commerce domain. We argue, therefore, that the solution to these problems could be a novel concept: *dynamic electronic institutions*, which are the main object of this research.

There is no significant work in the literature on dynamic electronic institutions: the notion was recently introduced as a challenge for agent-based computing, when the term *dynamic electronic institution* appeared in a roadmap for agent technology [7].

As we will explain in this paper, which is a summary of our recent work [9], we believe that dynamic electronic institutions arise from the convergence of two lines of research: electronic institutions and coalition formation. In this work we explain the main limitations of these two areas to justify the need to develop dynamic electronic institutions. We explore these ideas and concepts, present some first approaches, and introduce our first exploratory work.

This paper is organized as follows. In section 2 and 3 we explain the most important characteristics of coalition formation and electronic institutions respectively, their key concepts and their main problems, in order to show how mutual benefit could arise by combining them. Next, section 4 illustrates the connection points between coalition formation and electronic institutions, and suggests that a combined approach of these two areas could be beneficial for both. Section 5 addresses the concept of dynamic electronic institution and its lifecycle. In section 6 we explain the foundation phase, which we introduce as the process of turning a coalition into a temporary electronic institution. Section 7 shows our first exploratory work (in the area of Operations Other Than War, OOTW). Finally section 8 presents conclusions and future research.

2 Coalition Formation

Forming coalitions between self-interested agents is a subject of long-standing interest for game theory researchers, and more recently for computer science researchers (particularly in multi-agent systems, MAS).

The coalition formation problem arises when a group of agents needs to work together to achieve tasks in an environment. The main idea behind coalition formation is that cooperation between agents may be useful for the group even if the agents are selfish and try to maximize their own outcomes. Mutual profit can be gained from sharing resources and redistributing tasks. Coalitions allow agents to perform tasks together that they would be unable to perform individually.

At its simplest the problem can be defined as in [5]: *"Given a population P of agents and a list of tasks or goals T, select subgroups of agents S1, S2, S3… of P to address each of the tasks in T"*.

In the general case this problem is computationally intractable (NP-Hard). This is very easy to understand: given N agents and k tasks, there are $k(2N\text{-}1)$ different possible coalitions. The number of coalition configurations (different partitions of the set of agents in coalitions) is of the order $O(N^{(N/2)})$ [16]. Therefore, it is clear that an exhaustive search of the coalition configuration space is not feasible (when the number of agents is large). In [14] this problem has been diminished by using the idea of alliances which effectively reduce the search space.

In its beginnings, coalition formation was addressed in game theory. However the game theoretic approach is centralized, and as stated before, computationally intractable. Recent works in distributed artificial intelligence have resulted in distributed algorithms with computational tractability. Some of these recent and interesting studies are: *dynamic coalition formation* [5] and *iterated coalition formation* [8].

Shehory presents several issues that have to be addressed by agent designers when they attempt to provide their agents with coalition formation mechanisms [16]:

- Given a set of tasks and a set of agents, which coalitions should an individual agent attempt to form?
- Once the agent knows which coalitions it would prefer to form, what mechanisms can it use to form the coalitions?
- Given a specific coalition formation mechanism, what guarantees of efficiency and quality are provided?
- Once a coalition has been formed, how should its members go about distributing the work/payoff?
- At times, it may be necessary to provide means for the agents to dissolve the coalitions they have formed.

In our opinion there are three important problems that appear when the coalition is formed. We argue that these problems can be studied and possibly solved by turning the coalition into a temporary electronic institution (dynamic electronic institution), which is the main aim of our research. These three problems are:

- *Teamwork problem*: this problem arises from team coordination between distributed agents. A serious problem in coordinating distributed agents that have formed a coalition, is how to enable them to work together as a team towards a common goal. This is a strategy coordination problem.
- *The Work/Payoff distribution problem*: once a coordination strategy has been chosen there is a problem of how the work and the payoff should be distributed among the agents.
- *Emergent behaviour problem*: coalition formation approaches have an agent-centred view so the emergent behaviour of the global system can become chaotic and unexpected.

3 Electronic Institutions

From a social point of view, it is easy to observe that the interactions between people are often guided by institutions that help and provide us with structures for daily life tasks. Institutions structure incentives in human exchange (political, social, or economic). Institutions establish laws, norms and rules to respond to emergencies, disasters, et cetera. Somehow we could say that institutions represent the rules of the game in a society or, more formally, are the humanly devised constraints that shape human interaction [12].

The idea to use organisational metaphors to model systems was early proposed [13, 19]. These approaches suggest structuring the agent society with roles and relationships between agents. But the study of electronic institutions is a relatively recent

field (the first approach was [11]). The main idea is simple, and it could be summarized by imagining groups of intelligent, autonomous and heterogeneous agents, which play different roles, and which interact with each other under a set of norms, with the purpose of satisfying individual goals and/or common goals. As a first impression, it could seem that these norms are a negative factor which adds constraints to the system, but in fact they reduce the complexity of the system, making the agents' behaviour more *predictable*.

Research in Distributed Artificial Intelligence (DAI), and more specifically, research in MAS, has focused on the individual behaviour of agents (agent-centred view). But this agent-centred perspective is not useful in complex systems like *open agent systems*, where their components (agents) are not known a priori, can change over time, and can be heterogeneous and exhibit very different behaviours. Open agent systems are also characterized by limited trust and conflicting individual goals. In these kinds of systems, this vision that is focused on the agent can cause the emergent behaviour of the global system to be chaotic and unexpected. In critical applications this can be a significant problem, and it is evident that is necessary to introduce regulatory measures which determine what things the agents can do, and what they cannot. It is here where the institutions acquire importance. Agent-centred approaches can be useful for closed and small systems, but they fail to design open systems [2, 15].

In Noriega's thesis [11], an abstraction of the notion of institution is introduced for the first time. He is also the first to use the term *agent-mediated electronic institution*, which he describes as: computational environments which allow heterogeneous agents to successfully interact among them, by imposing appropriate restrictions on their behaviours.

Continuing and extending the ideas of Noriega's thesis, there is Rodríguez-Aguilar [15] who emphasizes the need for a formal framework which allows to work with general electronic institutions.

From these first approaches to this area, to the actual lines of research, there have been different European research groups working on similar subjects, each one with its particular perspective and approach to the problem. At the moment, many efforts are dedicated to this research area. The proof of this is that in 2003, five PhD theses intimately related to this subject were presented. The theses are: [1, 3, 4, 6, 18].

These different approaches to electronic institutions have demonstrated how organisational approaches are useful in *open agent systems*, but in our opinion, they still have several problems and limitations. We have summarized these problems in the following list:

- All the approaches to electronic institutions are based on medium to long-term associations and dependencies between agents. This characteristic is useful in some application domains but it is a significant problem in other domains, where changes in tasks, in information and in resources make temporary associations necessary (short-term electronic institutions).
- Electronic institutions require a design phase (performed by humans). It is necessary to automate this design phase in order to allow the emergence of electronic institutions (without human intervention) in open agent systems.

- Agents can join and leave institutions, but how do these entrances and exits affect the institutions' norms and objectives? Could these norms and objectives change over time?
- When an institution has fulfilled all its objectives, how can it dissolve its components (agents)?

In our opinion, these problems and limitations can be studied and possibly solved with a coalition formation approach to electronic institutions, in order to develop dynamic electronic institutions. This is the main objective of our research. In the next section we compare coalition formation and electronic institutions in order to find the connection points between them.

4 Connection Points Between Coalition Formation and Electronic Institutions

Firstly, to show how coalition formation and electronic institutions can converge, we have outlined some of their features in Table 1.

Table 1. Comparison between coalition formation and electronic institutions

	Coalition formation	**Electronic institutions**
Design/Creation process	Emergent association (without human intervention)	**Designed by humans**
Duration, length	Short to medium-term associations	**Long-term associations (static)**
Effects of agents' entrances/exits (adaptivity)	DCF methods (Dynamic Coalition Formation [5])	**Not contemplated**
Dissolution process	DCF and iterated coalition formation mechanisms [8]	**Not contemplated**
Teamwork	**Distinct methods (not solved yet)**	Directed through roles/norms
Work/Payoff distribution	**Distinct methods (not solved yet)**	Directed through roles/norms
Complexity (# group configurations)	**NP-Hard, $O(N^{(N/2)})$ [16]**	Limited by the designer and by norms
Openness	Tested on open agent systems	Tested on open agent systems
Local/Global agents' vision & mission	Local vision & local-global mission	Local-global vision & local mission
Typical applications	Game theory, e-commerce, rescue operations, etc.	E-commerce, auctions, e-health, etc.
Metaphors	Teams, alliances, clans, swarms	Institutions/Organisations

Table 1 plots some important features of coalition formation and electronic institutions. Some of the features are limitations and problems that we have commented on in previous sections (we have highlighted them in the table).

In our opinion the limitations of electronic institutions could be reduced by using a coalition formation approach to the problem. In this way the creation and dissolution processes could be automated through emergent associations (in relation to the agents' objectives) and emergent dissolutions (when the associations have fulfilled all their objectives). Furthermore, a coalition formation approach could improve the institutions' adaptivity processes, basically in relation to the agents entering and exiting the system, and could allow short to medium-term associations (a very interesting characteristic in domains where temporary associations are required).

Moreover, we believe that some of the problems of coalition formation (teamwork and work/payoff distribution) could be reduced and possibly solved by using a joint vision of coalition formation with electronic institutions, that is, by turning coalitions into institutions (we explain this process in the next sections). In this way, coordination between agents is improved because the agents acquire roles and obligations through the institution's norms, which reduce the system's complexity by defining exactly what things the agents can do, and what they cannot.

Therefore, we argue that combining coalition formation and electronic institutions is beneficial for both areas. Our proposal is to study a coalition formation approach to electronic institutions in order to develop dynamic electronic institutions, which from our point of view is a novel and interesting challenge for agent-based computing.

There is no previous significant work on dynamic electronic institutions: this idea has just recently been introduced as a challenge for agent-based computing. It first appeared when the term *dynamic electronic institution* appeared in a roadmap for agent technology [7]. In this roadmap the authors emphasize the need to study these kinds of institutions and state:

"The next challenge for agent-based computing is to develop appropriate representations of analogous computational concepts to the norms, legislation, authorities, enforcement, etc., that can underpin the development and deployment of dynamic electronic institutions. (…) Virtual organisations involve dynamic coalitions of small groups that can provide more services and make more profits than an individual group. Moreover, such coalitions can disband when they are no longer effective. At present, coalition formation for virtual organisations is limited, with such organisations largely static. The automation of coalition formation will save both time and labour, and may be more effective at finding better coalitions than humans in complex settings."

5 From Coalitions to Institutions: Dynamic Electronic Institutions

We argue that Dynamic Electronic Institutions (DEIs from now on) can be described as follows: emergent associations of intelligent, autonomous and heterogeneous agents, which play different roles, and which are able to adopt a set of norms in order to interact with each other, with the aim of satisfying individual goals and/or common goals. These formations are dynamic in the sense that they can be automatically formed, reformed and dissolved, in order to constitute temporary electronic institutions on the fly. This type of institution should be able to adapt its norms and objectives dynamically in relation to its present members (agents).

There are several application domains that require short-term agent organisations or alliances, in which DEIs could be applied. Table 2 shows three application domains for DEIs.

Table 2. Table 2 shows application domains for dynamic electronic institutions, and also shows the different types of agents, associations, norms and dynamics for each domain

Application Domain / Scenario	Description	Types of agents	Required short-term associations	Required regulatory measures
B2B: Business to Business Electronic Commerce	In the B2B scenario, enterprises try to fulfil the new market requirements by connecting themselves to *temporary corporations* with flexible structures that can change dynamically. **Scenario dynamics:** market opportunities.	Enterprises, Organisations	Corporations or alliances (virtual organisations)	Electronic Contract (rights, duties and penalties)
OOTW: Operations Other Than War	In this case the idea is to form coalitions of groups of people, non-governmental organisations (NGO's), institutions providing humanitarian aid, and official governmental initiatives. These coalitions should become institutions in order to carry out humanitarian operations like peace-keeping, disaster relief operations, etc. **Scenario dynamics:** disasters, emergencies, etc.	NGOs, groups of volunteers, hospitals, etc.	Short–term associations for solving concrete emergencies	Obligations and prohibitions for the different types of agents
MANETs: Mobile Ad-Hoc Networks	Ad-hoc networks are characterized by a dynamic topology and self-organizing capacity. The scenario requires dynamic, flexible and short-term associations between nodes. **Scenario dynamics:** topology, connections, etc.	Nodes: mobile hosts or devices (PDAs, Cellular phones, etc.)	Temporary nets	Self regulatory organisation
Other possible scenarios: nano-satellite constellations, grid-computing, semantic web.				

In our opinion DEIs should have a lifecycle made up of by three phases: Formation, Foundation and Fulfilment (We call this lifecycle "3F cycle" [9]). Figure 1 depicts this cycle.

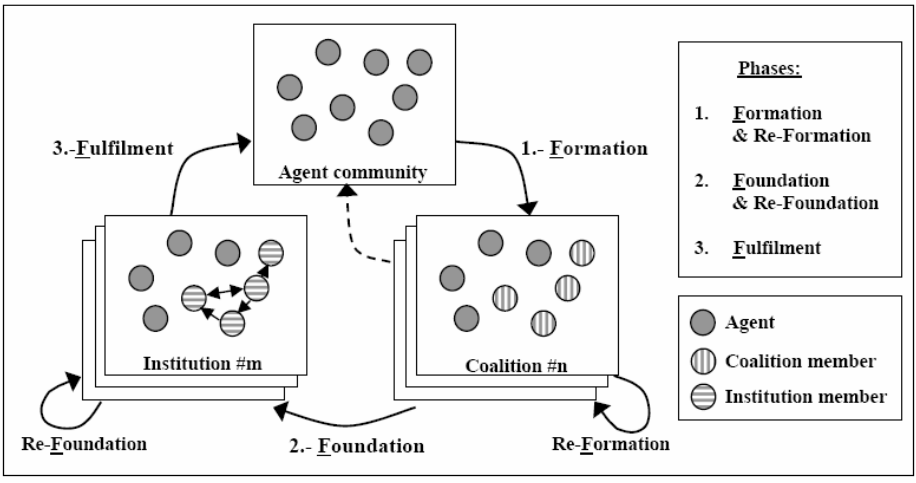


Fig. 1. The lifecycle of Dynamic Electronic Institutions (3F cycle): Formation, Foundation and Fulfilment

- *Formation phase*: this is the coalition formation phase. Associations between agents which have the same (or similar) goals emerge. Other notions such as trust between agents should also be considered as important factors in the coalition formation phase. A coalition formation mechanism (protocol and strategies) is necessary to allow agents to form coalitions.
- *Foundation phase***:** the process of turning the coalition into a temporary electronic institution. This phase is the real challenge, because the process of turning the coalition into a temporary electronic institution is not a trivial problem. We address this question in the next section.
- *Fulfilment phase***:** when the institution has fulfilled all its objectives, the association should be dissolved. This phase occurs because the association is no longer needed, or because the institution is no longer making a profit. A dissolution process can also be considered after the formation phase.

Within this lifecycle there are also the *re-formation* and the *re-foundation* processes as shown in Figure 1. The *re-formation* process facilitates reconfiguring the coalition when member changes occur, and the *re-foundation* process facilitates reconfiguring the institution when member changes occur.

One of these three phases has been poorly studied in the past: the foundation phase (and the *re-foundation* process). At this moment, we are focusing our work on this phase. In the next section we try to explain how this complex process could be carried out.

6 The Foundation Phase

We define foundation as the process of turning a coalition into a temporary electronic institution. This phase is a real challenge because the process of turning the coalition

into a temporary electronic institution is not a trivial problem. It requires the agents to adopt a set of norms that regulate their interactions. This must be an automated process, without any human intervention, so agents must be able to reason and negotiate at a high level.

Our perspective on this problem is that to construct an institution from zero without human intervention may be too difficult, so we argue that an approach based on using knowledge from previous cases (like Case Based Reasoning, CBR) could be interesting and useful for solving this issue (also for the *re-foundation problem*). Presently, we are directing our efforts in this direction.

The foundation phase can be represented as a black box in which a coalition is turned into an institution (Figure 2). In our approach we would like to work with the IIIA's model of an institution [3] so that we can use EIDE tools (Electronic Institution Development Environment [17]), which in our opinion are the most complete and functional tools for this purpose. Therefore, in our approach, the output of the black box for the foundation phase should be an ISLANDER specification of the institution. Thus, AMELI could be used for running the institution specified with ISLANDER [17].

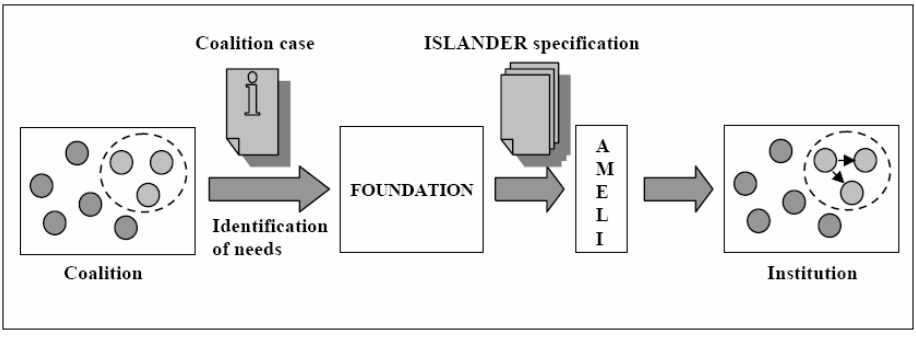


Fig. 2. The Foundation Phase represented using a black box

In our approach we are using Case-Based Reasoning. CBR is based on the idea that new problems are often similar to problems that have been encountered previously and that past solutions may be of use in the current situation.

A stored case refers to a problem situation and contains a description of a problem, and its solution, and a new case contains the description of the problem to be solved. Case-based reasoning is a cycle. There are four phases in the process: Retrieve, Reuse, Revise and Retain.

With a CBR approach to the foundation process, when a coalition has been formed and needs to turn itself into an institution, agents should consult their case databases in order to find the stored institution's specification that adapts best to the present situation, and should then make the pertinent reforms to the selected specification in order to obtain an institution that works correctly.

The CBR process should be done in a distributed way (each agent has its own stored institution cases, in relation to its own experience in the system) or in a centralized way (there is a central database with the stored institution cases).

Our first steps are based on a centralized CBR approach. We have used the JADE/Agent-0 framework [10], and our agents have a BDI architecture with a mental state composed of three mental categories: *Beliefs*, *Commitments* and *Capabilities*. Within our CBR mechanism, when a coalition has been formed and needs to turn itself into an institution, agents consult a centralized case database in order to find the stored institution's specification that adapts best to the present situation. Then the agents have to adopt the norms specified in the selected institution in order to turn the coalition into an institution. In our system, norms are adopted by taking on new commitments. Figure 3 shows a scheme of this process.

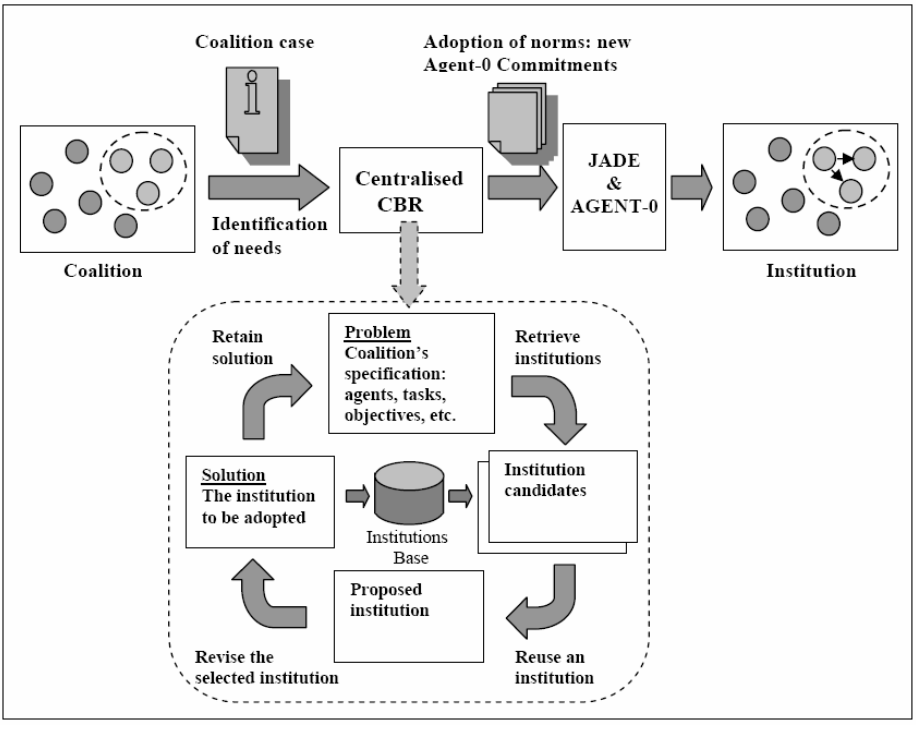


Fig. 3. A scheme of the foundation process within our system

7 Exploratory Work: Operations Other Than War

The task of planning humanitarian aid operations is a challenging problem because it means that a large number of different types of organisations have to collaborate in order to solve problems in fast-changing environments. OOTW are characterized by non-hierarchical decision making and decentralized control. OOTW involves different types of operations: non-combatant evacuations, disaster relief operations, food distribution, etc. The *agents* involved in these operations can be: groups of volunteers, NGO's, institutions, official governmental initiatives, etc.

In order to test our system we have started with a very simple scenario which involves three types of agents:

- *Volunteers*: every agent represents a group of volunteers.
- *Doctors Without Borders (DWB)*: every agent is a unit of this international humanitarian aid organisation.
- *Hospitals*: every agent represents a hospital.

In our scenario there is a succession of disasters, each of which causes a different number of wounded. The agents attempt to form coalitions in order to aid and heal the wounded, and these coalitions can attempt to form institutions with the aim of improving their teamwork.

Within a coalition, each agent acts in its own way in order to fulfil the coalition's objectives, whereas within the institution there are norms that regulate the agents' actions and interactions in order to improve the teamwork and reduce the complexity of the system. This means the agents' behaviour is more predictable.

Each agent has been implemented with the Agent-0 language and has a mental state with beliefs, commitments and capabilities [10]. The agent's behaviour is governed by commitment rules, which define the agent's character in relation to the environment and other agents. At the moment we assume that agents share an OOTW ontology, but in the future agents will have to assimilate the ontologies through the adopted institution, in order to face specific problems.

The coalition that is formed generates a *coalition case* that is a specification of the coalition and contains: the number of agents, the number of different roles, the distinct types of roles, and the coalition's objectives. Then, a centralized CBR mechanism searches for the specification of the stored institution that adapts best to the present *coalition case.*

To initialize the system some institution specifications must be introduced into the case base before starting up the system. Therefore, in the first CBR iterations the coalitions can reuse previous institution cases.

We have been carried out two experiments with our system and this scenario:

- First experiment uses the JADE/Agent-0 framework in the OOTW scenario. The system starts with a population of 12 agents: 5 volunteer groups, 5 DWB units, and 2 Hospitals. There is a succession of disasters and agents can attempt to form coalitions. The coalition formation mechanism is very simple and based on grouping together the agents that share the objective: *heal wounded*. The coalition formation process has time limits, and every disaster causes a different and random number of victims. We have obtained an average of 55.38% saved victims, and an average of 9.8 agents per coalition.
- Second experiment is the same as the previous one, except that institutions can be founded. The system also starts with a population of 12 agents: 5 volunteer groups, 5 DWB units, and 2 Hospitals. Then, there is a succession of disasters, and agents can attempt to form coalitions. In this experiment coalitions can turn themselves into institutions using a centralized CBR mechanism (the foundation phase mechanism). The coalition formation mechanism is the same, and each disaster also causes a different and random number of victims. We have introduced 4 *institution cases* into the centralized base to initialize

the system and facilitate first institution adoptions. In this case, we have obtained an average of 56.7% saved victims, and an average of 9.2 agents per coalition. This is a small improvement with respect to the previous example, but it is not significant. We have worked with our own framework, our own scenario, and our own examples, so we need to compare our results with other platforms and systems in order to validate them. The importance of these experiments is to show that the foundation phase is feasible, and that the DEI lifecycle can be fully implemented.

8 Conclusions and Future Work

We have presented the notion of *dynamic electronic institutions* (DEIs), which from our point of view arise from the convergence of two lines of research: electronic institutions and coalition formation. We have introduced and conceptualized their lifecycle: 3F cycle (Formation, Foundation and Fulfilment).

One of these three phases has been poorly studied in the past: the foundation phase. At this moment, we are focusing our work on this phase. In this paper, we have presented our CBR approach to the Foundation Phase.

We are using the JADE/Agent-0 framework (modelling agents' norms as commitment rules) [10], and we are working on the OOTW scenario (Operations Other Than War).

We have presented our first experiments and results, which are encouraging but not yet significant; they show that the foundation phase is feasible, and that the DEI lifecycle can be fully implemented.

There are several open issues in DEIs. These include works on the institutions' adaptivity and on the dissolution process (fulfilment phase).

References

1. Dignum, V.: A model for organizational interaction. Based on Agents, Founded in Logic. Ph.D. Thesis, Utrecht University (2003).
2. Esteva, M., Rodríguez-Aguilar, J. A., Sierra, C., Garcia, P., Arcos, J. L.: On the Formal Specification of Electronic Institutions. Lecture Notes in Artificial Intelligence, Vol. 1991, Springer-Verlag, pp. 126-147 (2001).
3. Esteva, M.: Electronic Institutions: From specification to development. PhD thesis, Universitat Politècnica de Catalunya (2003).
4. Fornara, N.: Interaction and communication among autonomous agents in multiagent systems, Ph.D. Thesis, University of Lugano (2003).
5. Klusch, M., Gerber, A.: Dynamic coalition formation among rational agents. IEEE Intelligent Systems, 17(3):42-47 (2002).
6. López y López, F.: Social power and norms. Impact on Agent Behaviour. Ph.D. Thesis, University of Southampton (2003).
7. Luck, M., McBurney, P., Preist, C.: Agent Technology: Enabling Next generation Computing. A Roadmap for Agent Based Computing. AgentLink II (2003).

8. Mérida-Campos C., Willmott, S.: Modelling Coalition Formation over Time for Iterative Coalition Games. In proceedings of EUMAS'04 (2nd European Workshop on Multi-agent Systems), Barcelona (2004).
9. Muntaner-Perich, E.: Towards Dynamic Electronic Institutions: from coalitions to institutions. Thesis proposal submitted to the University of Girona in Partial Fulfilment of the Requirements for the Advanced Studies Certificate in the Ph.D. program in Information Technologies. Girona (2005).
10. Muntaner-Perich, E., del Acebo, E., de la Rosa, J. Ll.: Rescatando Agent-0. Una aproximación moderna a la Programación Orientada a Agentes". CEDI'05 (I Congreso Español de Informática), Taller de Desarrollo en Sistemas Multiagente: DESMA'05. Granada (2005).
11. Noriega, P.: Agent Mediated Auctions. The Fishmarket Metaphor. Ph.D.Thesis, Universitat Autònoma de Barcelona (1997).
12. North, D. C.: Economics and Cognitive Science. Economic History 9612002, Economics Working Paper Archive at WUSTL, (1996).
13. Pattison, H. E., Corkill, D. D., and Lesser, V. R.: Distributed Artificial Intelligence, chapter Instantiating Descriptions of Organizational Structures, pages 59-96. Pitman Publishers (1987).
14. Pechoucek, M., Barta, J., Marík, V.: CPlanT: Coalition Planning Multi-Agent System for Humanitarian Relief Operations. Multi-Agent-Systems and Applications: 363-376 (2001).
15. Rodríguez-Aguilar, J. A.: On the design and construction of Agent-mediated Electronic Institutions. Ph.D. thesis, Universitat Autònoma de Barcelona (2001).
16. Shehory, O.: Coalition Formation: Towards Feasible Solutions. Proceedings of International Central and Eastern European Conference on Multiagent Systems. Prague, Czech Republic, Springer-Verlag Heidelberg: 4-6 (2003).
17. Sierra, C., Rodríguez-Aguilar, J. A., Noriega, P., Esteva, M., Arcos, J. L.: Engineering Multi-agent Systems as Electronic Institutions. Novática, 170, July-August (2004).
18. Vázquez-Salceda, J.: The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains. The HARMONIA framework. PhD thesis, Universitat Politècnica de Catalunya. Artificial Intelligence Dissertation Award, ECCAI, (2003).
19. Werner, E.: Distributed Artificial Intelligence, chapter Cooperating Agents: A Unified Theory of Communication and Social Structure, pages 3-36. Pitman Publishers (1987).

Institutionalization Through Reciprocal Habitualization and Typification

Eric Baumer[1] and Bill Tomlinson[2]

[1] University of California, Irvine, Department of Informatics, 127B Computer Science Trailer, Irvine, CA 92697-3430
ebaumer@uci.edu
[2] University of California, Irvine, Department of Informatics, Department of Drama, and Arts Computation Engineering (ACE) Program, Computer Science Bldg, Room 403A, Irvine, CA 92697-3440
wmt@uci.edu

Abstract. When constructing multiagent systems, the designer may approach the system as a collection of individuals or may view the entire system as a whole. In addition to these approaches, it may be beneficial to consider the interactions between the individuals and the whole. Borrowing ideas from the notion of social construction and building on previous work in synthetic social construction, this paper presents a framework wherein autonomous agents engage in a dialectic relationship with the society of agents around them. In this framework, agents recognize patterns of social activity in their societies, group such patterns into institutions, and form computational representations of those institutions. The paper presents a design framework describing this method of institutionalization, some implementation suggestions, and a discussion of possible applications.

1 Introduction

There are many approaches to developing multiagent systems. Agent-oriented software engineering takes a top-down approach, decomposing the problem domain into components that can be addressed by individual agents or groups thereof [12]. In artificial life, a bottom-up approaches focuses on overall properties emerging from the interactions of individual agents [16]. It may be beneficial to focus not only on the individual the whole, but rather on the interactions between the individual and the whole. Social psychology has taken a variety of perspectives on the individual's relationship to the society [5, 18]. As an example of applying these different approaches, Dillenbourg, et al. [7] present a description of how changing the unit of study along the spectrum of individual to group changes the nature of educational research. This paper presents a technique for applying some of these concepts to multiagent systems, allowing the agents in the system to form computational representations of institutionalized actions.

Social psychology examines the nature of the individual's relationship to its society. Different theorists have taken a number of different approaches in trying to describe the nature of this relationship between self and society. Mead argues that the

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 122 – 134, 2006.

self has no meaning outside of society, that the self only exists with any meaning as a response to and interaction with the generalized other [18]. Cooley introduced the notion that the self is reflected in society in what he called the reflexive self or the looking-glass self, arguing that our self-concept comes only from the impressions we get about ourselves based on others' actions toward us [5]. Goffman posits that our social form is based on the manner in which we present ourselves [10]. Berger and Luckmann argue that the individual exists in dialectic relationship with society, and through this dialectic exchange the individual defines his or her own social reality at the same time that society is defining the individual [3]. More recent work synthesizes some of these earlier approaches. For example, Tice and Wallace present a case that even the reflected self is distorted by the lens of self-presentation, so that we see ourselves not as others actually see us but as we think others see us [22]. Although the specifics of these approaches differ, they all place the individual in an exchange of mutual sense making with society, where the society and the individual are simultaneously trying to understand one another.

This paper focuses on the work of Berger and Luckmann, specifically the social construction of reality [3]. This approach places the individual in a dialectic relationship with society in which the individual experiences society both as an objective and a subjective reality. With society as an objective reality, the individual experiences society as a unified whole, an external entity that dictates the individual's social reality. With society as a subjective reality, the individual internalizes society, making sense of it by relating it to his or her own subjective experiences, and then acts on that subjective interpretation. By experiencing society through this dual nature, individuals can reciprocally typify patterns of each other's actions and interactions to form institutions, which describe the behavioral patterns present in that society. This concept is related to some previous work that allows synthetic characters to form context-specific emotional memories [23]. The present framework, however, places those memories in a broader institutional context rather than the context of interaction with a specific individual. There has also been previous work has been done in the area of electronic institutions [9]. This work uses electronic institutions as a way to specify the interactions and commitments between different agents in an open environment [6, 21, 24]. However, all these approaches focus on using institutions as a method of specifying agent behavior to assist in the creation of multiagent systems. The work described here uses institutions to recognize and reason about emergent patterns of actions in multiagent systems.

This paper presents a framework for institutionalization that applies these concepts to multiagent systems, provides some aspects of an implementation, describes difficulties and limitations that might arise in that implementation, and discusses possible applications in domains such as human-machine interaction, societal or behavioral studies, and analysis of multiagent systems.

2 Social Construction

As stated above, Berger and Luckmann's social construction approach places the individual in a dialectic relationship with society [3]. In this dialectic relationship, the individual experiences society both as an external, objective reality and as an internal,

subjective reality. Through these multiple experiences of society, society acts to define the individual, while the individual simultaneously acts to define society. "Society is a human product. Society is an objective reality. Man is a societal product." (p. 61) [3]

It is important to note that these three moments of definition are indeed three distinct moments, but they are also three simultaneous and cyclical events. The events of the dialectic relationship between the individual and society are continuous and coincident. This work focuses on the individual's experience of the society both as objective reality and as subjective reality in the process of forming institutions.

2.1 Society as Objective Reality

One component of the individual experience is that of society as objective reality, where in the individual experiences society as an external entity. Society dictates certain norms and patterns of behavior in the form of institutions, to which the individual must subscribe. "The institution posits that actions of type X will be performed by actors of type X." (p. 54) [3] The institution serve to dictate which types of actors take what types of actions. However, it also functions to give context to those prescriptions, so that actions of type X are performed by actors of type X in the context of other types of actors performing other types of actions. These patterns of actions form institutions that dictate to the individual the patterns of action that are to be taken within the society.

Consider the example of a courtroom trial. One would not expect to see the bailiff overseeing the court atop the bench, the lawyers sitting in the jury box listening, and the judge transcribing the events. The institution of a trial dictates different roles for different types of actors in the context of a trial. By knowing about an institution, actors in a situation know how to behave, and an individual can make sense of a situation. This leads to the individual forming its own impression of the scene.

2.2 Society as Subjective Reality

The other component of individual experience is that of society as subjective reality, wherein the individual goes through a process of sense making about the social occurrences around him or her. The individual attempts to understand societal activities based on his or her previous experiences, arriving at his or her own subjective interpretation of society. The individual then acts based on this subjective interpretation, thus influencing society. The individual's actions are then experienced as objective reality by other individuals.

When the individual takes actions based on his or her subjective reality, the individual is essentially presenting that subjective version of reality as an objective reality to other members of the society. The subjective reality is then confirmed or denied by the continued actions of others. Only when an individual's subjective reality is confirmed by the actions of others, when an individual has the same conceptualization of its society as others do, is that individual is a member of the society.

3 Framework

The main focus of this paper is to present a framework for institutionalization among autonomous agents based on some of these ideas from social construction. This framework builds on previous work in synthetic social construction [2], but rather than designing for a specific application this paper presents a general framework that can be applied to forming institutions in a wide range of multiagent systems. This section describes the components of the framework and how they interact to allow agents to form institutions via their interactions.

3.1 Habitualization

Habitualization[1] is the process by which actions that are frequently repeated with the same temporal relationships to one another are cast into a pattern. These patterns form groups of actions, so that an agent can reorganize its assessment of its own actions to treat these groups of actions individually or as a single unit. Any pattern of actions repeated at least once is habitualized to some degree, with those actions performed more frequently having a higher degree of habitualization. Habitualization is a non-social process by which the agent recognizes the habituality of its own actions and thus requires no interaction with other agents. This is not to say that an agent does not habitualize actions it takes in conjunction with other agents. Quite to the contrary, habitualization is more beneficial in the company of other agents, as it works hand-in-hand with typification, described below.

3.2 Typification

Typification is a process similar to habitualization, but rather than recognizes internally occurring patterns of behavior as in habitualization, typification recognizes externally occurring patterns enacted by other agents. Typification allows an agent to observe other agents and determine what types of actions are typical of what types of agents, and what types of agents typically perform what types of actions. Just as any pattern of actions repeated at least once is habitualized to some degree, any observed pattern of actions creates some degree of typification. Typification is most important in that reciprocal typification, which entails two agents typifying each other's actions, combined with habitualization leads to the formation of institutions.

3.3 Institutionalization

In the context of multiagent systems, an institution is a computational representation of a certain behavioral pattern implicitly agreed upon through the actions and interactions of the members of a society to which the institution applies. Institutions are formed by means of institutionalization.

[1] This is not to be confused with habituation, wherein repeated exposure to a specific stimulus decreases the degree or frequency of the associated response.

Institutionalization is the process by which multiple agents habitualized their own actions and reciprocally typify each other's actions. Agent A habitualizes its own actions and typifies agent B's actions, while agent B is habitualizing its own actions and typifying agent A's. When agent A's typification of agent B's actions matches angent B's habitualization of its own action and vice versa, an institution is formed. Alternately, an agent can form an institution by observing and typifying the actions of other agents without actually participating in their interactions. The institution crystallizes the repeated patterns of actions composed of actions taken by the various agents involved in those patterns. It gives context to individual habitualizations and typifications by placing them in conjunction with other habitualizations and typifications. These institutional patterns of actions describe the types of interactions undertaken by the agents in a system.

Institutions serve a number of purposes. First, they allow agents to form computational representations of patterns of behavior. Those patterns may be analyzed by the agent itself or reported to human designers wishing to examine the nature of interactions in a system. Second, they provide a social learning mechanism. If a given agent observes other agents performing institutionalized patterns of actions, the agent may be able to learn those patterns of actions and participate in such patterns at a later time. Third, their predictive power can be used to inform action selection. Given a number of possible institutions that might fit the current events in which the agent is participating, the agent can choose its actions based on the institution that has the most beneficial outcome or leads other agents into a desirable course of action.

Institutions serve to abstract away from the differences between specific occurrences to generalize about all instances of that pattern of actions. One important way in which they do this is by defining roles, sets of actions taken within the pattern of some institution.

3.4 Roles

Roles abstract away the differences between individual actors that act in specific situations to generalize about all actors that might perform a set of actions in a given institution. An agent fulfilling a role in an institution is not only "a particular actor performing an action of type X, but" is demonstrating a "type-X action as being performable by *any* actor" (p. 72) [3] who fits the description of the role. This generalization does not pigeonhole any actor who fits this description into only performing type-X actions in this context. Rather, it represents a trend that applies to most but not necessarily all agents and situations. The specific description of a given role is the set of other roles a performer of the given role is likely to assume. For example, actors who perform role P in one institution are likely to also perform role Q in some other institution, so it is a reasonable conclusion that an agent performing role P will likely perform role Q, as well.

4 Implementation

Thus far, this paper has described a framework to allow agents in multiagent systems to form institutions based on habitualized and typified patterns of actions. This section

proposes an implementation outline for that framework and discusses what will be required for such an implementation.

4.1 Pattern Recognition

The key component to habitualization and typification is the ability to recognize patterns of actions that are repeated frequently with the same or similar temporal relationships to one another. While there has been much research done on pattern recognition, many of the algorithms are complex and were developed for some specific application domain, such as image, face, and object recognition; audio processing; or speech recognition. Current pattern recognition algorithms are not amenable to the patterns of actions and interactions being recognized in this framework.

A number of approaches could be taken to the implementation of this component. One approach could combine some type of reinforcement learning [15], such as Q-learning [25], with various prediction methods to determine which events frequently occur within temporal proximity to what other events. Another approach could be to seed a pattern recognition algorithm with arrangements of certain types of events. In this way, the pattern recognizer does not have to start from scratch but rather has a platform of known patterns upon which to build other patterns. However, this method requires a specific set of pre-made patterns and is not as open-ended in the patterns of behavior it will recognize. A third approach could be to form a pattern from any set of events that occur within close temporal proximity. Once a pattern has been formed, the agent can determine whether or not the given pattern is an accurate or effective description for the events in its environment. Methods for making this determination are discussed below in the section on pruning.

4.2 Comparing Institutions

The ability to compare a series of currently occurring events to a known set of institutions acts as a key component to this framework. It is this ability that allows an agent to determine if some known institution is being enacted, what agents are performing which roles in this manifestation of the institution, and what the given agent's role in the institution might be. There are a number of aspects that will be used to accomplish this.

Below is described how institutions can be seen as a hierarchy, where large, complex institutions are described as compositions of smaller, simpler institutions. This very naturally gives rise to a tree-like structure for the representation of institutions, possibly in a method similar to the one described by Alspaugh [1]. Even institutions that are composed only of simple events may be seen to have a tree structure, where the institution is the root and the events that compose it are the leaves. Comparing two institutions could be accomplished in part be calculating a tree-to-tree distance algorithm [17] based on the tree-like representations of those institutions.

However, institutions are described by more than the events that compose them. They are also described by the temporal relationships between those events. Ordering the branches in the tree left-to-right based on the temporal order in which the events begin gives a starting point for the tree-to-tree distance calculation. However, the temporal relations between each pair of events must also be examined, since two

institutions having the same tree structure may still be significantly different based on the temporal relations among their constituent events.

One more aspect must be compared to determine the degree to which a series of currently occurring events matches an institution. The actors performing the roles in the current manifestation of the institution must be compared to the characterizations of those roles. As stated above, a given role is characterized by the other roles that actors performing the given role are likely to take on. So for each role, there is a list of other roles, each with an associated probability indicating how likely a performer of the current role is to perform any one of a number of other roles. A similar list of roles and associated probabilities is maintained for each agent. When an agent is compared against a role it is performing, these probability matrices are compared, and their difference is the difference between the given agent and the role it is performing. However, there is a problem of scalability here, because each agent must carry around a probability matrix for each other agent of which it is aware. It may be possible for these matrices to decay over time, so that after periods with little interaction two agents no longer keep detailed information about each other's probability matrices. Similarly, the agent could perform pruning, so that rather than removing probability matrices at some given time interval they are removed based on the number and frequency of interactions with some other agent.

Another significant difficulty lies in determining to which institutions to compare the currently occurring events. Since this comparison will be a complex and likely computationally expensive process, minimizing the actual number of full comparisons will help keep the system scalable. One way to do this would be a heuristic determining which institutions to check based on their frequency of occurrence. The most commonly occurring institutions will be checked first, because the time spent checking them will yield a match much more often than the rarely occurring institutions. Along these lines, the heuristic could also incorporate a starting sequence, a series of events that always occurs at the beginning of a manifestation of the institution. Comparing currently occurring events against this starting sequence would be far less computationally intense than comparing the entire institution and would allow the full gamut of institutions to be checked rather than just the most frequently occurring ones.

4.3 Hierarchical Structure

Once a series of events has been recognized as a manifestation of an institution, those events can either be treated individually or as a whole. If treated as a whole, this manifestation of an institution can be seen as a single event, one that could possibly combine with others to form another higher-level institution. In this way, large, complex institutions can be formed from combinations of shorter, simpler institutions.

It may also be beneficial to organize not only institutions but individual events into a classification hierarchy. For example, running, walking, swimming, and crawling are all types of locomotion. In certain institutions, the method of locomotion should not matter as much as in others. In an institution such as the transportation of materials, it does not matter if an agent is using bipedal walking, caterpillar treads, or wheels, just that it is moving from one place to another while carrying some material. Using a super-event from which all these other sub-events inherit would allow any of

the sub-events to be used as an instance of the super-event. This enhances the flexibility of institutions and reduces the total number of institutions needed, since a single institution can describe an action that involves any type of locomotion rather than needing a separate institution for each type.

4.4 Pruning

In this framework, pruning is analogous to the maintenance of subjective reality, wherein the individual verifies his or her subjective reality by acting on it and then determining if his or her actions are comprehensible to others. Institutions are monitored to determine how accurate they are as predictors and how frequently they are manifest. Institutions that are inaccurate predictors or infrequently manifest are either removed from the agent's collection or decreased in the frequency with which they are checked. However, this creates a possible difficulty for special purpose institutions. For example, the institution of a wedding occurs very infrequently, and so we do not walk around on a daily basis trying to understand our interactions in terms of a wedding. However, when we see a wedding, whether expected or unexpected, we have little trouble recognizing it as such. This example again brings up some of the challenges in selecting which institutions to compare against the currently occurring series of events.

5 Limitations and Difficulties

Some of the limitations and possible difficulties in this institutionalization framework were mentioned in the implementation section above. This section discusses two specific problems that may arise in the framework's implementation and suggests some possible approaches to addressing these difficulties.

5.1 Bootstrapping

One of the features listed above was that of a hierarchical structure, that is, that institutions can be composed of other institutions. At the lowest level, institutions are composed of single simple events. However, if an agent is not aware of any institutional structures, only of simple events, how does it bootstrap from simple events into basic institutions?

One approach to solving this, mentioned above, is to seed the agent with some basic institutions, on top of which the agent can construct other institutions. Using this approach limits the institutions that an agent can recognize to those that are based on institutions foreseeable by the designer. However, it gives more control and direction to the type of institutions that will be formed.

Another possible method would be giving the agent a specific list of all the possible simple events in its environment and using a pattern recognition algorithm to determine when these events occur repeatedly with the same temporal relationships to one another. As mentioned above, this would require complex and flexible pattern recognition, but it would allow for more variability in the sorts of institutions that an agent could recognize. For example, events A, B, and C might frequently occur in close temporal proximity and always with the same temporal relations to one another,

but if the agent is only looking for patterns of events A and C as dictated by the designer, the agent would not be able to recognize this new pattern of events and would not learn the institution that might lie behind those events.

Both of the above approaches are possible solutions to the problem of bootstrapping if the agent is in an environment in which simple events occur in a discrete and recognizable way, such as market simulations, auction environments, or operations other than war scenarios. However, what occurs if an agent does not even have knowledge of what events are possible in its environment? In a simulation or virtual environment, it is possible for an agent to have full knowledge about what events are possible within that environment before actually interacting with it. Work is being done on activity recognition in open physical environments but has not yet been perfected in a general purpose form. Further details of this problem are discussed below.

5.2 Granularity

In very constrained environments, it would be possible to give an agent a specification of what comprises an event. However, in an open environment, this is significantly more complex. Consider an agent on a desktop computer observing the actions of the user. For simplicity's sake, suppose that the only information about the user available to the agent is the input from the mouse and keyboard. In this case, what constitutes a single event? One press of a key? One click of the mouse? Any continuous mouse movement in the same direction? Any set of inputs received within a specific temporal proximity? Figuring out exactly what comprises an event is a non-trivial matter.

Here it is possible to see the utility of having institutions that are composed of other institutions. In this example, we can allow simple repeated patterns of inputs to be learned as institutions. It may not make immediate sense to classify such activities as institutions, e.g., the institution of double clicking. However, groupings of these low-level institutions may be grouped into higher-level patterns, such as opening a file, cutting-and-pasting, or turning off the machine. In this way, institutions could be developed from the ground up. Such an approach could be applied to agents interacting with users in the physical world. Basic sensory-motor level actions could be used to form basic institutions, which would then comprise higher-level institutions that deal with interactions with users or aspects of the world.

6 Discussion

Up to this point, this paper has presented a framework for applying some of the concepts of social construction to multiagent systems, allowing them to form institutions in an adaptive and non-predetermined way. This section will discuss aspects of how this framework relates to other aspects of autonomous agents research.

Many multi-layered architectures have been proposed for the design of intelligent systems. In at least two of these [19] [20], there is a reflective layer that monitors the activities of the sensory-motor and deliberative layers. It is in this reflective layer that the above described framework would be implemented. As is apparent, the framework is not designed for motor-level action control, but rather to monitor actions and

interactions in order to form patterns describing those interactions. Once such patterns are formed, they may be referenced by the sensory-motor or deliberative to assist in guiding future actions.

Another interesting question is that of embodiment or embodied action, which has application not only to autonomous agents but also to the broader fields of artificial intelligence, human-computer interaction, and many other disciplines. Traditionally, embodiment has been used to describe the way an agent is situated in its physical environment [4]. In recent years, embodied action has come to incorporate not only an agent's physical environment but also its social setting and how these aspects of its environment change over time [8]. While there has been a good deal of work in the field of artificial intelligence on embodiment in the physical sense, there has not been as much work focusing on embodiment in the social sense. The framework brings the concept of embodied interaction to bear on multiagent systems, giving agents a means to represent and reason about their social environment and the changes occurring therein.

There are also several possible approaches in deploying this framework, depending on the purpose of the system. The framework was designed to be deployed in such a way that each agent in a system has its own set of institutions based on its subjective experiences of interacting with other agents. These institutions could be compared to determine if all agents in an interaction approach that interaction in the same way. Another possibility is setting up an observational system independent from all the agents' interactions. This observer would determine what institutions are present in a system without actually interacting with it. However, this would require some modification, because the observer is never required to act based on the institutions it forms and thus has no way of verifying the validity of these institutions. An alternate verification here could be checking how well the observer's institutions predict actions of other agents, then modifying the institutions based on the accuracy or inaccuracy of these predictions.

7 Applications

There are several possible ways in which institutionalization could be applied to multiagent systems. These include strategy negotiation, so that negotiated strategies do not have to be constantly renegotiated; determining societal norms and then finding behavioral outliers in societies; and user modeling to discover and facilitate commons usage patterns. Three possible applications are discussed in detail below.

7.1 Human-Machine Heterogeneous Systems

One of the main difficulties in deploying systems composed both of human and machine agents in the coordination of the two. This framework could provide a method for machine agents to learn patterns of interaction as institutions, facilitating interaction with human agents. Another similar application is human control of systems composed of heterogeneous agents. If the agents are built with specific tasks in mind, delegating tasks to them is not necessarily a complex task. For example, miner robots are given the task of mining some ore; courier robots are given the task of moving the

ore; etc. However, if the system is composed of heterogeneous, general purpose agents, this task becomes more difficult. A human operator must sort through the different types of agents, or worse examine each individual agent, determine which agents are suited for a particular task, and assign that task to all the suitable agents. Using the framework described here, a different approach might be taken. The human operator could take control of one general-purpose agent, one that he or she knows is fit for a certain task, and perform the given task. During the performance of the task, the agent's motor-level functions are controlled by the human operator, but its reflective layer is still monitoring its own actions, forming institutions that describe the tasks being performed by the operator. Once the task has been completed, control is returned to the agent, and the institutions learned by the reflective layer dictate the proper course of action. Furthermore, the computational representation of this institution could be supplied to all the agents in the system. Because the institution describes the actor in terms of an abstracted role, any agent suited for that role will perform it. Thus, the agents automatically determine which of them are fit to perform a certain task.

7.2 Agent Co-learners

Some work has already been done exploring the effect of artificial co-learners on the performance of human students [14]. However, the agents in these studies act in predetermined ways and do not adjust to the behavior of human users or other agents. The behavior and performance of co-learner agents has been shown to have a significant effect on both the performance of human learners and their attitudes toward their co-learners [14]. A co-learner could adapt to the human user's patterns of behavior, both attitudinal and performance-wise, so that the agent's behavior would be relative to that of the human user.

7.3 Analyzing Emergent Behavior

Emergent behavior in complex systems has been the topic of some discussion [11, 13, 26]. However, the presence and recognition of emergent properties are often subjective; it is up to the observer, often the system builder, to recognize when a pattern of behavior has emerged as a consequence of lower-level interactions. The framework described here could provide a method for determining when an emergent pattern has formed. Furthermore, treating lower-level actions and interactions as the basis for institutions, institutions formed in this framework could be analyzed to determine how and possibly why some lower-level interactions caused certain behavioral patterns to emerge.

8 Conclusion

This paper presents a framework that allows agents in a multiagent system to form institutions based on their behavioral patterns and build computational models of those thereof. Institutionalization, the process of forming institutions, results from reciprocal typification and habitualization among a group of agents. Within these institutions, the patterns of actions performed by agents are described in terms of roles, which

abstract away the differences between individual agents to describe which types of agents perform which types of actions and which actions are performed by what agents. Thus, the institutions form the basis for an agent's model of the interactions occurring in its environment. This institutionalization framework has broad possible applications, ranging from human-machine interaction, to user modeling, to analysis of multiagent systems.

References

[1] Alspaugh, T. Temporally Expressive Scenarios in ScenarioML *Institute for Software Research Technical Report UCI-ISR-05-06*, University of California, Irvine, 2005.

[2] Baumer, E. and Tomlinson, B., Synthetic Social Construction for Autonomous Characters. in *AAAI Workshop on Modular Construction of Human-Like Intelligence*, (Pittsburgh, PA, 2005).

[3] Berger, P.L. and Luckmann, T. *The Social Construction of Reality: A Treatise on the Sociology of Knowledge*. Irvington Publishers, Inc., New York, 1966.

[4] Brooks, R.A. Elephants Don't Play Chess. *Robotics and Autonomous Systems*, *6*.

[5] Cooley, C.H. *Human Nature and the Social Order*. Charles Scribner's Sons, New York, 1902.

[6] Cuni, G., Esteva, M., Garcia, P., Puertas, E., Sierra, C. and Solchaga, T., MASFIT: Multi-Agest System for Fish Trading. in *16th European Conference on Artificial Intelligence (ECAI 2004)*, (Valencia, Spain, 2004), 710-714.

[7] Dillenbourg, P., Baker, M., Blaye, A. and C, O.M. The evolution of research on collaborative learning. in Reimann, P. and Spada, H. eds. *Learning in Humans and Machine: Towards an interdisciplinary learning science*, Elsevier, Oxford, 1996, 189-211.

[8] Dourish, P. *Where The Action Is: The Foundations of Embodied Interaction*. MIT Press, Cambridge, MA, 2001.

[9] Esteva, M. Electronic Institutions: from specification to development *Artificial Intelligence Research Institute (IIIA)*, Technical University of Catalonia (UPC), Barcelona, 2003.

[10] Goffman, E. *The Presentation of Self in Everyday Life*. Doubleday, New York, 1959.

[11] Holland, J. *Emergence: from chaos to order*. Oxford University Press, Oxford, 1998.

[12] Jennings, N.R. On Agent-Based Software Engineering. *Artificial Intelligence*, *117* (2). 277-296.

[13] Johnson, S. *Emergence: the connected lives of ants, brains, cities, and software*. Simon & Schuster, New York, 2002.

[14] Ju, W., Nickell, S., Eng, K. and Nass, C. Influence of colearner agent gehavior on learner performance and attitudes *CHI '05 extended abstracts on Human factors in computing systems*, ACM Press, Portland, OR, USA, 2005.

[15] Kaelbling, L.P., Littman, M.L. and Moore, A.W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*. 237-285.

[16] Langton, C. *Artificial Life: an overview*. MIT Press, Cambridge, MA, 1995.

[17] Lu, S.-Y. A Tree-to-Tree Distance and Its Application to Cluster Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *1* (2). 219-224.

[18] Mead, G.H. *Mind, Self, and Society from the Perspective of a Social Behaviorist*. University of Chicago Press, Chicago, 1934.

[19] Murphy, R., Lisetti, C.L., Tardif, R., Irish, L. and Gage, A. Emotion-Based Control of Cooperating Heterogeneous Robots. *IEEE Transactions on Robotics and Automation*, *18* (5). 744-757.

[20] Norman, D., Ortony, A. and Russell, D.M. Affect and machine design: Lessons for the development of autonomous machines. *IBM Systems Journal*, *42*. 38-44.

[21] Sierra, C., Rodriguez-Aguilar, J.A., Noriega, P., Esteva, M. and Arcos, J.L. Engineering multi-agent systems as electronic institutions. *European Journal for the Informatics Professional*, *4*.

[22] Tice, D.M. and Wallace, H.M. The Reflected Self: Creating Yourself as (You Think) Others See You. in Leary, M.R. and Tangney, J.P. eds. *Handbook of Self and Identity*, The Guilford Press, New Your, 2003.

[23] Tomlinson, B. and Blumberg, B. AlphaWolf: Social Learning, Emotion and Development in Autonomous Virtual Agents. *First GSFC/JPL Workshop on Radical Agent Concepts. NASA Goddard Space Flight Center, Greenbelt, MD. January 2002. Published in Lecture Notes in Computer Science, Springer Verlag*, *2564*. 35--45.

[24] Vasconcelos, Esteva, M., Sierra, C. and Rodgriguez-Aguilar, J.A., Verifying norm consistency in electronic institutions. in *AAAI-04 Workshop on Agent Organizations: Theory and Practice*, (San Jose, CA, 2004), 8-14.

[25] Watkins, C.J.C.H. and Dayan, P. Technical Note: Q-Learning. *Machine Learning*, *8* (3-4). 279-292.

[26] Wolfram, S. *A New Kind of Science*. Wolfram Media, Champaign, IL, 2002.

On the Concept of Agent in Multi-robot Environment*

Stanisław Ambroszkiewicz and Krzysztof Cetnarowicz

Institute of Informatics, University of Podlasie, Al. Sienkiewicza 51, PL-08-110 Siedlce, Poland
and
Institute of Computer Science, University of Mining and Metallurgy, Al. Mickiewicza 30,
30-059 Krakow, Poland
sambrosz@ipipan.waw.pl, cetnar@agh.edu.pl

Abstract. Although, the software agent paradigm has been widely accepted, there is still a problem with situating (implementing) concrete agents in real environments. Agent is an abstract notion so that there is no straightforward mapping of real entities like robots, spacecrafts or any other autonomous systems to agents. An interesting discussion on this subject was done within the framework of LOGOS and ACT Agent Architecture in the context of ground and space systems, see [10,11]. In the paper we discuss the problem of situating agents in open systems consisting of cognitive heterogeneous robots that are supposed to perform jointly complex tasks. The starting point of the discussion is the archtecture of the M-agent.

1 Introduction

Multi-agent systems have been studied for more than ten years. However, a satisfactory technology for such systems creation has not been proposed yet. There are some approaches (see [6] for more details), that attempts to define a flexible framework for building multi-agent systems, but the research has not been completed, and the results are not always applicable for real problems.

An idea of M-agents that arises as an extension of the concepts of object, process and multi-model control systems ([4]) may be applied to development of distributed and decentralized intelligent systems. In the paper a proposal of a formal definition of the M-agent architecture, multi-agent world including a definition of agents, agents' space, agents' environment and the relations between agents and environment, and among agents themselves is presented.

The existing approaches (BDI architecture, Agent-0 etc., [8,9,6]) define some models of multiagent system, but according to the opinions of the authors there are some troubles with their applications. Some of the main inconveniences are listed below.

- They take (as a base of considerations) a low level of abstraction of the problem so that the proposed model of the agent architecture does not cover a wide range types of agents and multiagent systems.
- The formalism proposed is not always appropriate (generic enough) for real problems (with its specific features) to be represented in the formal model.

* The work was done within the framework of MNiI project No. 3 T11C 038 29.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 135–146, 2006.

The proposed idea of M-agents may be considered as an extension of the object, process and multi-model control systems concepts. It seems that it takes (as the starting-point) properly defined abstraction of the problem that can be a base for the Agent Oriented Analysis, Design and even Programming. In particular:

- The proposed approach makes possible to distinguish between a multiagent and a non-multiagent system, and keeps a track of the development process to build desired multiagent systems.
- The definition of the system consists of several levels of specifications. It makes possible to apply this approach to a wide range of MAS with simple as well as with complex agents.
- The proposed approach makes possible to use various formalisms (up to the choice of the designer) to make a more detailed specification of the system. It is reasonable for complex and large systems to be represented as multiagent systems.

The paper presents an introduction to the M-agent architecture and its application to the description of the multiagent systems. The proposed approach, gives methods and tools that make possible to define a multi-agent system for a given problem without using any specific programming language.

2 Informal Introduction to the Concept of M-Agent

Let us consider some problem for which a multiagent system (MAS) is to be created. To design a multiagent system and to build a multiagent model that solve the problem we have to define: living space for agents, agents of several types, and the relations between the agents and the living space as well as among the agents. The first step that must be undertaken to put in order the multiagent system (world) is to specify: what is an agent and what is not an agent. So the whole reality analysed from the point of view of the multiagent system may be divided into two parts (see Fig. 1): environment - that may be observed by the agent and represented by a corresponding model, and agent's mind - an area in which the agent builds and processes models of the environment. So we can consider that a given agent a remains and acts in an environment called V. For any created agent a the following notions are defined:

- strategies S (s - strategy, $s \in S$), goals Q (q - goal, $q \in Q$), and models of the environment M (m - model, $m \in M$).
- operation of the observation I, and operation of the strategy execution X.

The main idea of agent functioning is the following:

- The agent observes the environment and builds a model m of the environment in its mind. For this purpose it uses the observation (or imagination) operation I.
- The agent forecasts its possibilities using the strategy s. By applying the strategy s to the model m it obtains the model $m' = s(m)$ of the modified environment. Then, the agent compares (in its mind) the models m and m' using the function q that determines the goal of agent functioning. It serves to select the best (from the point of view of the goal g) strategy s^*.

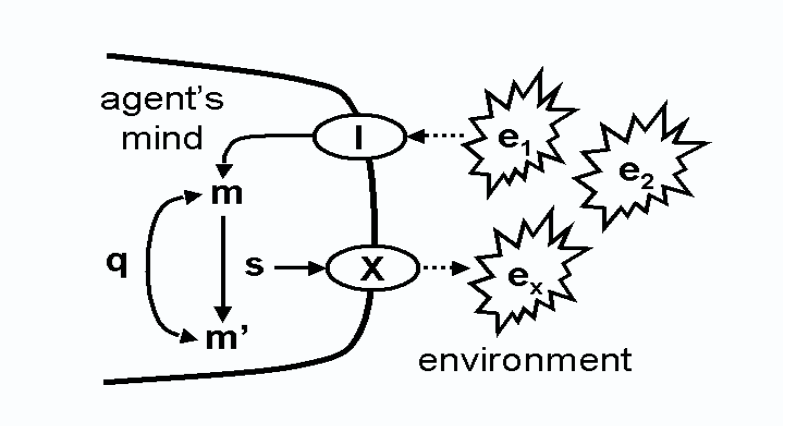


Fig. 1. Principle of the M-agent architecture. I - imagination operation, X - execution operation, m, m' - models of the environment, s - strategy, q - evaluation function, ev_1, ev_2 - events taking place in the environment, ev_x - event produced as the result of the realization of the strategy s.

– If the evaluation (comparison) of m and m' is satisfactory, the agent realizes the selected strategy s^*. The realization is represented by operation X - execution: $V' = X(s, V)$

Therefore, the first approach to the algorithm of a given agent may be expressed in three steps:

1. Observation - creation of the model m of the environment V: $m = I(M, V)$
2. Decision - selection of the optimal strategy s^*: $q(m, s^*(m)) = \max_{s \in S} q(m, s(m))$
3. Action - realization of the optimal strategy s^* in the environment V: $V' = X(s, V)$

The whole decision making (agent's reasoning) is carried out by processing the models of the environment in the area called agent's mind. We are unable to represent the "agent's mind" in the model m. It means that it is impossible that one may think how she/he is just thinking. The starting point to the agent oriented approach may be derived from the Object Oriented Analysis and Design and expressed in the following sentence: *"you are an agent, you must get along by yourself in the environment" ([6].* This is a useful starting point to define agents of the multiagent system within the framework of M-agent architecture.

More formal definition of the M-agent architecture is following. Let a denote an agent, and A denote a set of agents (called the agent configuration) that are in the MAS;($a \in A$). There may be several types of agents in the MAS. Then, a_i^g denotes an agent a_i of type g.

The definition of the MAS environment V is as follows. Let E denote a space of the MAS - living space for the agent of the system. An environment V is described by the triple:

$$V = (E, A, C) \tag{1}$$

where C - connection (binding) between agents A and agent space E. C may be defined as the location of agents in the space.

In some practical cases we may use a simplified version of the environment in which we consider only the part of the environment V that may be observed and modified by agent a. This environment is called "the environment from the point of view of the agent a" and is defined as:

$$V_a = (E, A_a, C_a) \tag{2}$$

where $A_a \subset A$ - a set of agents that may interact with the agent a, C_a - connection between agents A_a and the space E. The living space of agents may be defined by different ways suitable to a given application. We can consider a more complex model of M-agent architecture in which the agent is defined in the following way (see Fig. 2): Agent a, ($a \in A$) may be described as a 9-tuple:

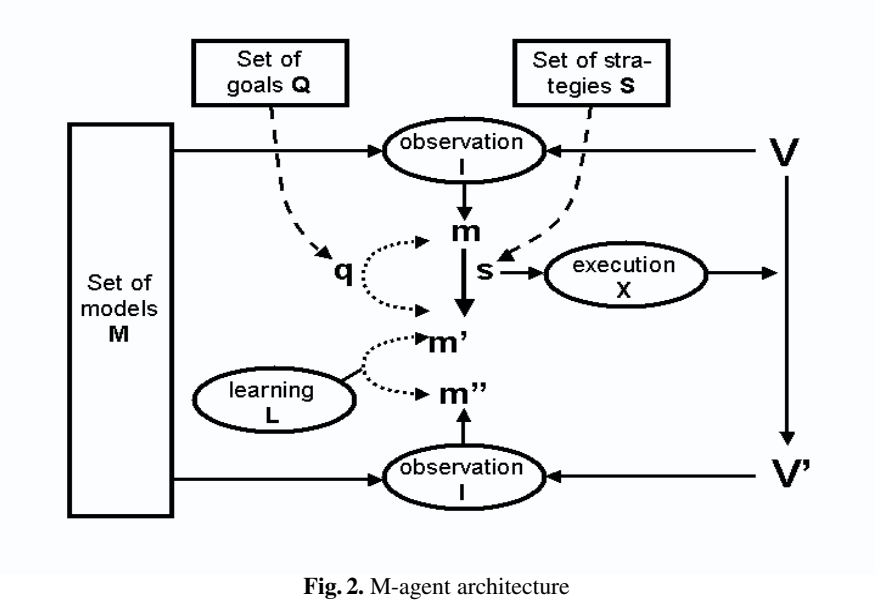


Fig. 2. M-agent architecture

$$a = (M, Q, S, I, X, L, m, q, s) \tag{3}$$

where L - agent's adaptation (learning) operation (see [6] for details), m - a model of the agent's environment, M - a set of models of the environment (configuration of models) representing agent's knowledge about the environment, ($m \in M$), and q denotes a goal of the agent a defined as follows.

$$q : M \times M \longrightarrow \Re\,, \quad q(m, m') \in \Re \tag{4}$$

where: m - an agent's model of the environment, m' - model of the (predicted) new modified environment, $\Re$ is the set of real numbers,

Q - a set (ordered) of agent's goals called configuration of agent's goals ($q \in Q$),
s - a strategy of the agent a ($s : M \longrightarrow M$, $m' = s(m)$), S - a set of agent's strategies called configuration of strategies, ($s \in S$),
I is agent's a observation operation defined as follows.

$$m = I(M, V) \tag{5}$$

X - agent's a strategy execution operation defined as

$$V' = X(s, V) \tag{6}$$

where $m \in M$, $s \in S$, $V = (E, A, C)$, V' - a new environment (of the agent a) created due to the execution of the strategy s by the agent a in the environment V.

The agent acts according to the following algorithm (see Fig. 2):

1. Start - the agent's creation:
   $(M, Q, S, I, X, L, m, q, s)$, and its location in the environment: $V = (E, A, C)$, then go to 2.
2. The agent looks around and builds the model of its environment: $m = I(M, V)$, then go to 3.
3. Selection of the best strategy s^* (that may be adopted and executed):

$$q(m, s^*(m)) = \max_{s \in S} q(m, s(m)) \tag{7}$$

   go to 4.
4. Execute selected strategy s^*: $X(s^*, V) = V'$; then go to 5.
5. The agent looks around and builds the model of its new environment: $m'' = I(M, V)$, then go to 6.
6. If there is considerable discrepancy between m' and m", then start an execution of learning operation L to realize adaptation to the environment. Next go to 3.

3 Agent's Model with Multiple Profiles

In some cases we may observe that beings alter (modify) their behavior as if they are changing their point of view. This modification may be considered as a new model of he M-agent described above. In such a case we may extend a model of an agent and consider it as multiprofile M-agent architecture. Each profile is an M-agent model. The agent observes the environment in each profile, and a final decision is made upon the results from each profile (see Fig. 3). Hence, we can consider an agent having several profiles:

$$a = (a_1, a_2, ...a_i...a_n) \tag{8}$$

where a_i is defined as:

$$a_i = (M_i, Q_i, S_i, I, X, L, m_i, q_i, s_i) \tag{9}$$

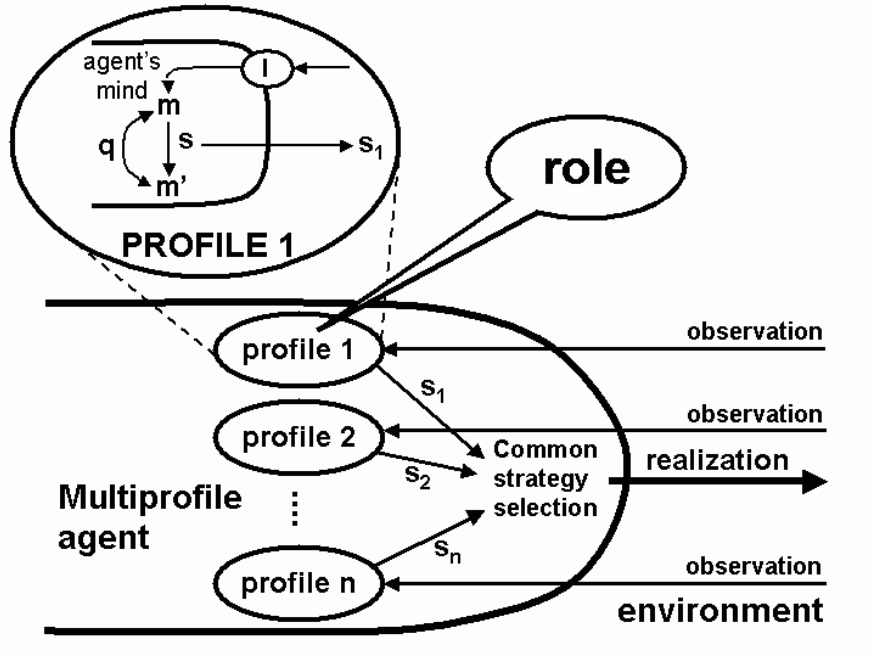


Fig. 3. A concept of the multiprofile M-agent architecture

The configurations M_i, Q_i, S_i are different and specific for each profile. For each profile (say i) its own strategy configuration S_i, models' configuration M_i, and goals' configuration Q_i have been defined. The operations I (observation), X (execution), L (learning) are common for the all profiles of the multiprofile agent.

The observation function I of the agent is defined as

$$(m_1, m_2, ...m_i, ...m_n) \;=\; I(M_1, m_2, ..., M_i, ..., M_n, V) \tag{10}$$

where for all $i\ (1 \leq i \leq n):\ \ m_i \in M_i$

In every profile the agent finds the optimal strategy s_i^*. The final decision to realize the strategy s is made by the agent with the use of the decision function U:

$$s = U(a_1, a_2, ..a_n, s_1^*, s_2^*, ...s_i^*, ...s_n^*) \tag{11}$$

The selected strategy s is realized by the agent in the environment: $V' = X(s, V)$

4 Planning

We can consider application of the proposed M-agent architecture to the planning operation preformed by agents.

Plan PL is defined as a sequence of strategies:

$$PL =< s_1, s_2, ...s_i, ...s_n > \tag{12}$$

where $\forall(1 \leq i \leq n)\ \ s_i \in S$.

Since strategies are function (from M to M), they may be composed. The composition $s \circ s'$ means the sequential execution of the strategies s', s. The plan PL is realized as a composition (sequential execution) of the strategies:

$$m_n = (s_n \circ s_{n-1} \circ ... \circ s_1)(m) \tag{13}$$

where m is the initial model of the environment whereas m_n is the final forecasted model of the environment after an application of the adopted plan PL (see Fig. 4). Having plan $< s_1, s_2, ...s_n >$, the agent may realize it in the environment. The realization may be defined in the following recursive way.

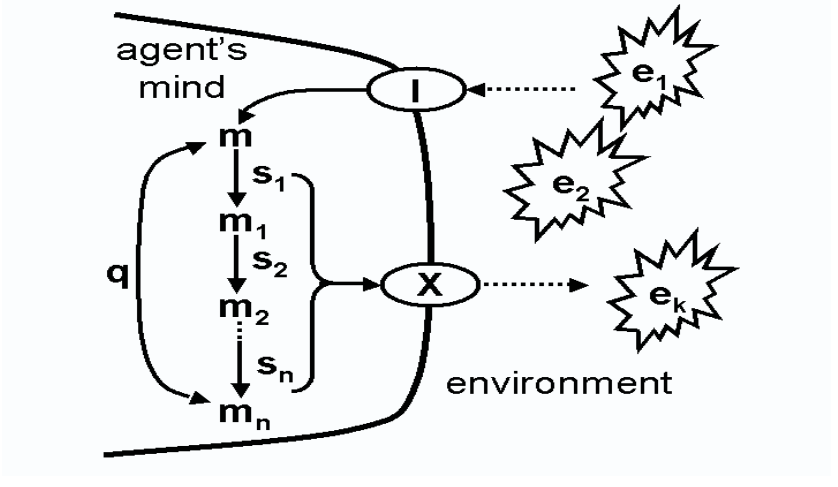


Fig. 4. Schema of a planning process in the M-agent architecture

$$X(V, < s_1, s_2, ...s_n >) = X(X(V, s_1), < s_2, ...s_n >) \tag{14}$$

Let V_n denote the result of the plan realization, i.e., the final state of the environment. After realization of the plan, the agent can build a model (m'_n) of the final environment V_n in its mind ($m'_n = I(M, V_n)$ using the observation function I. If the discrepancy between the forecasted model m_n and the observed model m'_n is not considerable, a new strategy s equivalent to the plan $< s_1, s_2, ...s_n >$ (i.e., $s = s_n \circ s_{n-1} \circ ... \circ s_1$) is created. This may be considered as a case of the learning function of the agent.

The planning operation may be enriched with the *milestones control system*. In this case the agent observes also intermediate results of the plan realization, not only the final state of the environment (see Fig. 4). So that the plan i.e. the sequence $< s_1, ...s_i, ...s_n >$ is provided with inserted milestones. The role of the milestone is played by marked strategies (for instance i). After execution of this "milestone strategies", the

agent observes the environment and builds its models m_i". Then, it compares the forecasted model m_i with the one obtained from the observation (m_i"). The discrepancy between these models serves to decide whether the plan is continued or stopped (i.e. modified or abandoned).

5 Negotiation

Using M-agent architecture we may analyze a negotiating process from the agent's point of view. We can distinguish between simple negotiations and more complex - planning negotiations.

In the case of simple negotiations (see Fig. 5) one agent - the agent "b" wants (taking its goal q_b into consideration) to perform its own strategy s_b. Agent "b" would like to have the agent's "a" agreement for that.

- For this purpose the agent "b", sending appropriate information, presents to agent "a" the state of the current environment (as it was observed by it, i.e., m_b) and the forecasted model of the environment resulting form execution of the strategy s, i.e., m'_b.
- The agent "a" using acquired information, builds in its mind the models $(m_b)_a$ and $(m'_b)_a$ which are its own version of models m_b and m'_b of the agent "b".
- The agent "a" evaluates the correctness of the proposed issue model $(m_b)_a$ and the changes in the environment using the forecasted model $(m'_b)_a$. For this purpose it uses its own goal q_a.
- If the changes are suitable for the goal of the agent "a" it agrees to the actions of the agent "b", if not, the agent "a" may prepare a new proposal for the agent "b" in the same way.
- An agreement closing negotiations is achieved when each agent accepts (taking its goal into account) the changes in the environment.

Agents may create a more complex solution using planning negotiations that may by carried out in the following steps (see Fig. 5):

- Agent "b" adopts strategy s_b that is convenient for the realization of its goal and sends information about it to agent "a".
- Agent "a", considering the proposed strategy $(s_b)_a$, tries to find its own strategy s_a, which, performed with the proposed strategy, is *good* from point of view of its goal q_a, and then informs the agent "b" about it.
- Agent "b" evaluates its own strategy s_b and the proposed $(s_a)_b$ using its own goal q_b. If the evaluation is successful the negotiations are completed with a construction of a common plan $< s_b,\ s_a >$. If not, the negotiation process is continued to find a new solution by means of new proposals given by the agents.

If the execution of both strategies (in a given order) is useful for the agents "a" and "b", the strategy is stored by the agents as a common new strategy realized in cooperation. This may be considered as an element of learning operation.

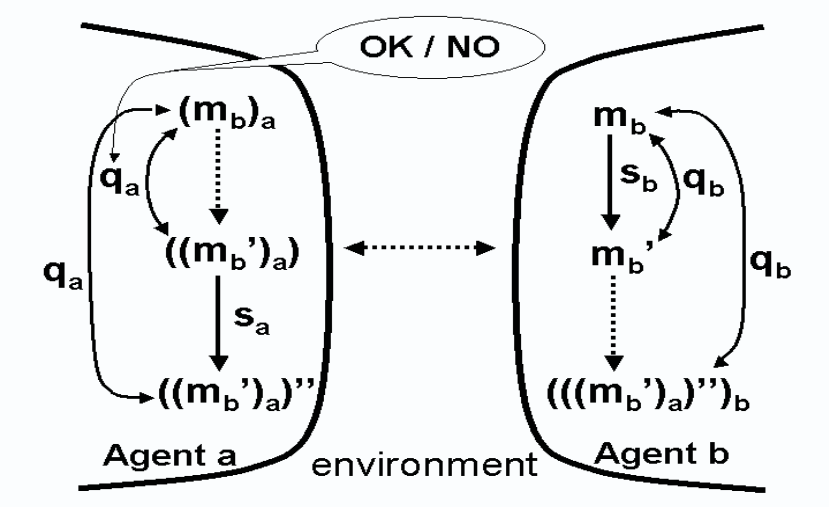


Fig. 5. Schema of the process of a simple negotiating process and schema of the process of a planning negotiation in the M-agent architecture

6 Agents and Robots

The proposed M-agent architecture may be used for the taxonomy of multiagent systems depending on the nature of the environment V. If the environment $V = (E, A, C)$ is a cyber-space, MAS contains software (possibly mobile) agents. If the environment is a real-space, there are devices (like mobile robots) that may communicate and interoperate. Designing MAS for such kind of environment is not easy. The immediate mapping of a robot into a single agent is not a good solution for complex robots that are supposed to perform sophisticated tasks in open, a priori unknown, environments.

Depending of the complexity of the models of the environment we may distinguish the following cases. If the environment is simple and fixed, then also agent can be simple and built according to the reactive architecture. If, however, the environment is complex, the agent must be cognitive so that its mind should according to the deliberative architecture.

A single agent may be associated to a single robot, see a) in the Fig. 6. However, the (software) agent need not to be installed on the robot's onboard computer. The agent may be situated in the cyber-space (as a process running on a remote host) controlling the robot from the host, see b) Fig. 6.

Hence, there are at least two possibilities. The first one is that robot is an intelligent autonomous being, and agent is its (optional) extension in cyber-space. The second one is that robot is an intelligent autonomous being, and an agent (or even a group of software agents) is its twin shadow in cyber-space processing the data from the robot and supporting robot's decision making.

Following this idea, we may say that robot (considered as autonomous intelligent being) is in the cyber-space as a software application (M-agent), whereas the real robot

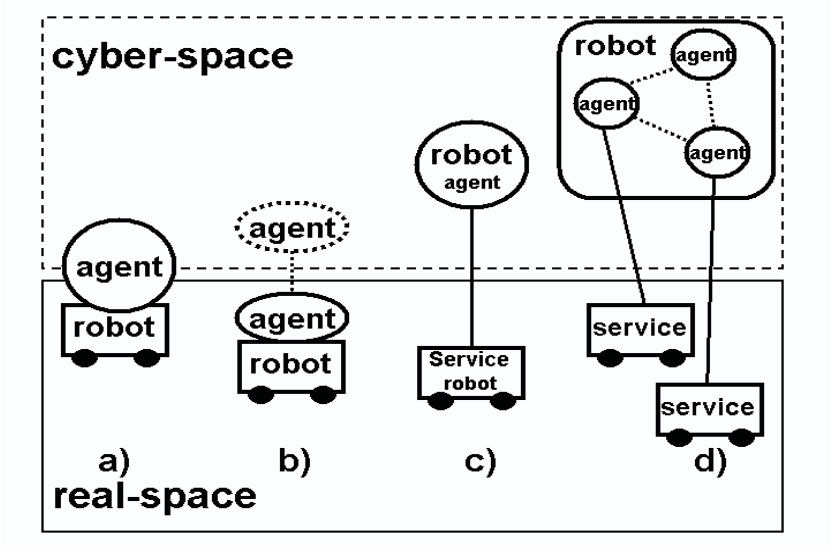


Fig. 6. Situating agents and robots in real-space or in the cyber-space

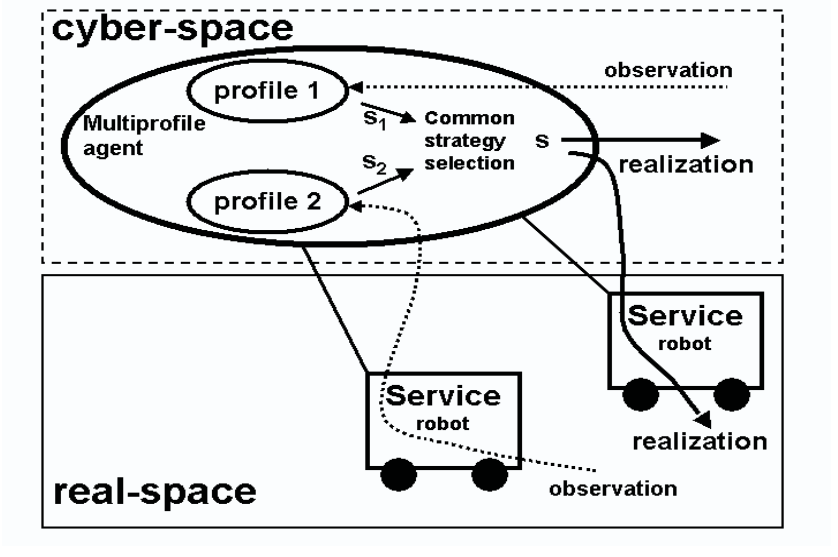


Fig. 7. Robots in the real-space as services for multiprofile M-agents in the cyber-space

is reduced to a device performing a specific service, and changing, in this way, the real-space, see c) Fig. 6. Moreover, a multiagent system in the cyberspace may use robots (as services) situating in the real-space, see d) Fig. 6.

Finally, multiprofile M-agent situated in the cyber-space may control a multi robot system, see Fig. 7. In this case each profile is responsible for controlling separate service performed by a device (robot) in the real-space.

For example, one profile is responsible for processing data coming from a camera of one robot, whereas the second profile is responsible for controlling the motion of the second robot. These two profiles are interrelated because one profile is responsible for controlling the robot that is scanning the environment (e.g., looking for suspicious and possible dangerous items), whereas the second profile is responsible for controlling the second robot that has ability do take a dangerous item and transport it to a safe place. Hence, there must be cooperation and coordination between these two profiles (consisting in selection and execution of a common strategy) in order to realize such a complex task.

7 Conclusions

Some new ideas concerning agent and multiagent systems in the context of robotics were presented. These ideas are based on our long term research and experiences in agent theory and technology, see [4,5,6] as well as in Web services and Semantic Web, see [1,2,3]. Although the ideas seem to be clear, there is still a lot to be done to provide a sound theoretical framework for them, and what is more important to create a technology that realizes and verifies them.

The limit of space does not allow to present details of our Service Oriented approach to robot and multi-robot architecture. This will be done shortly in our next paper.

There are ongoing implementations of the architectures of multi-robot system, presented in the previous section, as a part of the project *Interoperability of mobile and cognitive devices* supported by the MNiI grant No. 3 T11C 038 29.

References

1. Ambroszkiewicz S.: Agent Based Approach to Service Description and Composition. In Innovative Concepts for Agent-Based Systems: First International Workshop on Radical Agent Concepts WRAC 2002 McLean, VA, USA, January 16-18, 2002 Revised Papers. Springer LNCS 2564, ISSN: 0302-9743, pp. 135 - 149 (2003)
2. Ambroszkiewicz S.: enTish: An Approach to Service Description and Composition. Instytut Podstaw Informatyki Polskiej Akademii Nauk, Warszawa 2003, ISBN 83-910948-7-1, www site of the enTish project http://www.ipipan.waw.pl/mas/
3. Ambroszkiewicz S.: Entish: A Language for Describing Data Processing in Open Distributed Systems. Fundamenta Informaticae Volume 60, Number 1-4, April 2004, ISO Press, pp.41-66
4. Cetnarowicz K., Nawarecki E., and Zabinska M.: M-agent architecture and its application to the agent oriented technology", Proc. of the DAIMAS'97. International workshop: Distributed Artifcial Intelligence and Multi-Agent Systems, St. Petersburg, Russia, 1997
5. Cetnarowicz K., Kisiel-Dorohinicki M., and Nawarecki E.: "The application of evolution process in multi-agent world to the prediction system", Proc. of ICMAS96, AAAI Journal, Menlo Park, USA, 1996.
6. Cetnarowicz K.: *M-agent architecture based method of development of multiagent systems*. [w:] *Proc. of the 8th Joint EPS-APS International Conference on Physics Computing*, ACC Cyfronet, Krakw 1996

7. Nawarecki E., and Cetnarowicz K.: "Intelligent decision system with distributed reasoning", Proc. of the Eleventh IASTED International Conference Applied Informatics Annency, France 93, pages 152–155, IASTED Anaheim, Calgary, Zurich, 1993.
8. Rao A. S., and Georgeff M. P.: BDI-agents: from theory to practice. In Proceedings of the First Intl. Conference on Multiagent Systems. San Francisco, 1995
9. Shoham.Y.: AGENT0: A simple agent language and its interpreter. In Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), pages 704–709, Anaheim, CA, 1991.
10. Truszkowski W., and Rouff Ch.: New Agent Architecture for Evaluation in Goddard's Agent Concepts Testbed. NASA Goddard Space Flight Center. http://agents.gsfc.nasa.gov/papers/
11. Truszkowski, W., Hallock, H.: Agent Technology from a NASA Perspective. NASA Goddard Space Flight Center. http://agents.gsfc.nasa.gov/papers/

An Approach for Autonomy: A Collaborative Communication Framework for Multi-agent Systems

Warren R. Dufrene Jr.

PhD Candidate, Nova Southeastern University
Ft. Lauderdale, Florida
dufrene@nsu.nova.edu

Abstract. Research done during the last three years has studied the emersion properties of Complex Adaptive Systems (CAS). The deployment of Artificial Intelligence (AI) techniques applied to remote Unmanned Aerial Vehicles has led the author to investigate applications of CAS within the field of Autonomous Multi-Agent Systems. The core objective of current research efforts is focused on the simplicity of Intelligent Agents (IA) and the modeling of these agents within complex systems. This research effort looks at the communication, interaction, and adaptability of multi-agents as applied to complex systems control. The embodiment concept applied to robotics has application possibilities within multi-agent frameworks. A new framework for agent awareness within a virtual 3D world concept is possible where the vehicle is composed of collaborative agents. This approach is considered for application to the complex tetrahedron structure system from NASA Goddard Space Flight Center (GSFC) developed under the Autonomous Nano Technology Swarm (ANTS) program.

1 Introduction

This paper describes the development of an approach to apply a CAS Agent-based virtual framework to a model of the NASA Goddard Space Flight Center (GSFC) tetrahedron structure developed under the Autonomous Nano Technology Swarm (ANTS) program and the Super Miniaturized Addressable Reconfigurable Technology (SMART) architecture program [1]. The above projects, directed under Dr. Steven Curtis, represent an innovative set of novel concepts deploying adaptable, self-organizing structures composed of many tetrahedrons. The current technology program is pushing current applied Agents Concepts to new levels of requirements and adaptability. The proposed research discussed within this paper is an independent effort conducted through Nova Southeastern University (NSU) for a dissertation research effort to be conducted by the author under direction of Dr. Jim Cannady. The proposed research work is being undertaken because there is a need to advance the understanding of autonomy and the control and communications for multi-agent systems applied to complex systems.

The GSFC TETwalker (Figure 1) will be modeled using the Open Dynamics Engine (ODE) Software Development Kit (SDK) for simulation of the mechanical

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 147 – 159, 2006.

properties of the TETwalker structure [2]. The generic modeling of the structure will then be incorporated into a virtual 3D graphics simulation. This simulated environment will provide the testing ground for application of CAS techniques meshed with the proposed framework.

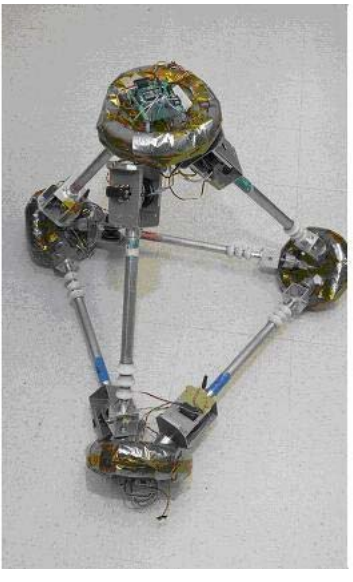


Fig. 1. NASA TETwalker prototype assembly that will provide the rough model prototype http://t2www.nasa.r3h.net/images/content/111408main_tetwalker_4.jpg

The achievement of autonomous systems has mainly concentrated on individual systems such as robots, ground vehicles, Unmanned Aerial Vehicles (UAV's), underwater vehicles, and software agents [3], [4]. Autonomous multi-agent systems have remained difficult to implement to date due, in part, to the increases in complexity over a single complex system [5]. Single autonomous systems have often been complex in nature and research continues in order to bring enhancement and autonomy to those systems [6], [7], [8]. The dynamic environment and external forces that affect these complex systems have contributed to the complexity of these systems and the obtainment of autonomy in these systems [9], [10].

1.1 Simplifying the Agent Approach

Standard applications of agent technology involve agents that must be adaptable, intelligent, and interactive with the environment where the agent software is applied. This level of growth in agent capabilities has often evolved to requirements of a complex system within itself. The agents must possess more advanced sensory inputs to meet the requirements of dynamic environments and the agents must also possess an increased computational intelligence capability not easily achieved given the current state of microprocessor horsepower.

The proposed research will address the issues of complex systems from the perspective of CAS. This perspective looks at the entire world of the multi-agent's interaction. This includes the multi-agent body (robot, TETwalker, vehicle, etc.), the dynamic physical world the agent body must operate in, and the interrelation between the agents and that environment. Drawing from previous work on the obtainment of knowledge (Figure 2) the agent will hold a degree of relationship with itself, other agents, and eventually overall behaviors and goals.

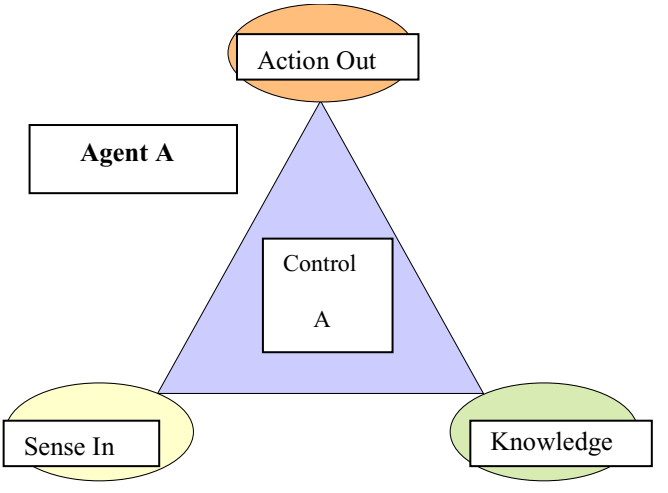


Fig. 2. Standard Agent Control with integration of knowledge "sensing" gained from past performance (Dufrene 2004)

The relationship held plays an important role in the communication architecture that will be developed in the proposed framework. To emulate the basic structures discussed in current CAS research the agents must be built with as little intelligence as possible. The data passed to each agent will adjust and contribute to the Sense, Rules, and Action sequences in order to collect what we can call knowledge (Figure 3). Information shared with higher order agents must incorporate the communication channels to allow the passing of data to the agents that would react to the information and thus cause the action required or desired. If done successfully the framework would include a relationship between the lower agents and the 3D virtual world that could allow the vehicle to converge on an imaginary point in the virtual world much like Particle Swarm Optimization (PSO) conversion, Cellular Automata (CA), or fractal conversion. The benefit of such a framework would allow the virtual spot to also exist within the vehicle itself or represent an agent node. The degree of membership between the Agents and the environment, system, tasks, and other Agents will change as the TETwalker interacts with its structure and environment (Figure 4).

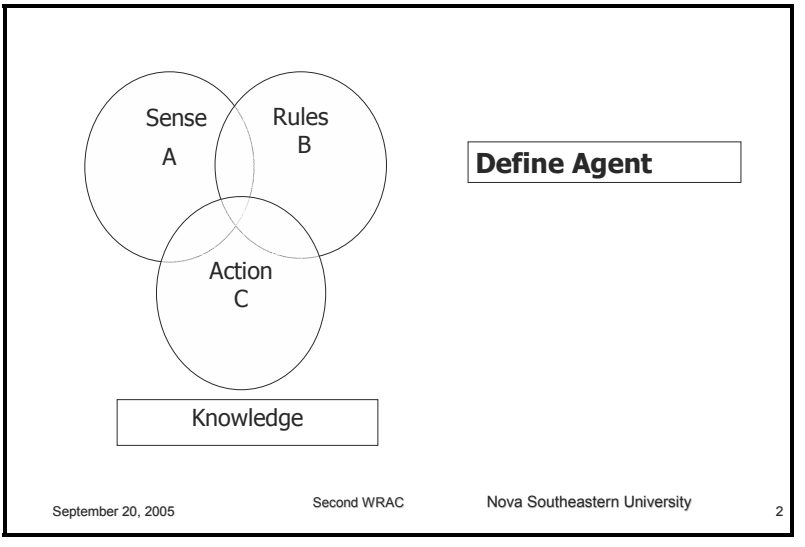


Fig. 3. Venn diagram denoting knowledge building incorporated in information and data passed to Agents and their definitions

The CAS techniques studied have presented a common core element that involves the emersion of CAS properties. That element is the inclusion of randomness and the degree of randomness. This research will investigate the effects of randomness as part of the equation of CAS properties. As in nature and biological colonies, the random interactions that take place in CAS play an important role in evolving properties that contain higher degrees of capabilities than the initial components of the system.

1.2 Benefits of Simplifying the Agent Approach

Why move beyond typical control schemes? A typical controls problem benefits from the use of proportional control, integral control or Proportional Integral Derivative (PID) control methods. An Adaptive Control Scheme is used to allow one or more of the control parameters to change in real time, which allows for adjustment to limited dynamic conditions. These types of control schemes have excellent reliability and predictive behavior when operating under their designed control margins. All of the typical control schemes require the same reductionism techniques used in science today in order to successfully model control parameters. This requires the system being reduced into smaller parts that are controllable and solvable. Complex systems that require adaptation to environments that are dynamic will require increased capabilities and real time adjustment of control margins. If you will, these future complex agents systems will require a Goal Oriented Distributed Intelligence (GODI). It is conceivable that CAS may have properties that are beneficial to the development of a GODI architecture.

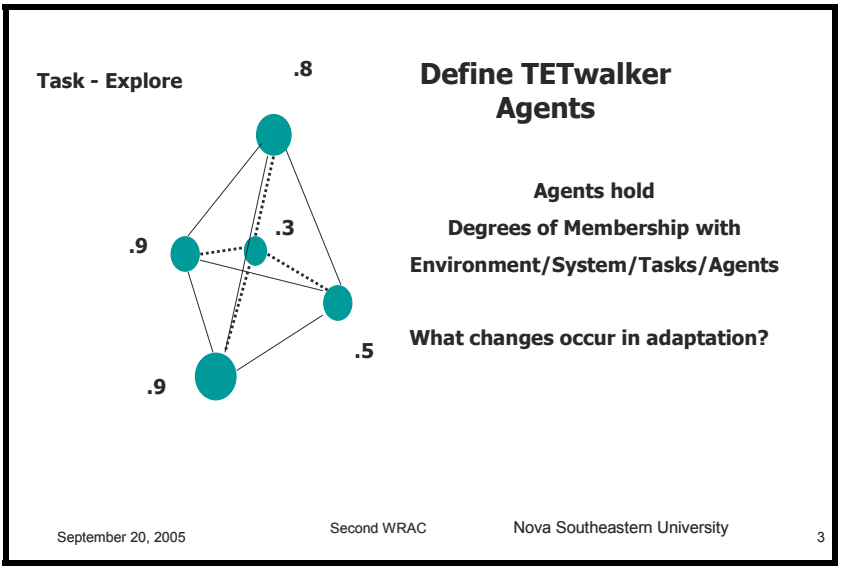


Fig. 4. Definition of TETwalker Agents and Degrees of Membership with Environment/System/Tasks/Agents

Some of the benefits of applying CAS techniques are the emersion of a higher order of capability, adaptation, self-organization, self-configuring possibilities, and an evolution of control through the use of simple agents. This process exists in nature and biological processes. The struggle in CAS research application is predicting the emerged property or even the property desired. It is suggested that the development of this framework can provide a novel control strategy that can be applied to the GSFC TETwalker that allows the growth and adaptation to multi-tetrahedrons through relationships with each agent and the virtual dynamic world built for simulation. These agents would require a simpler intelligence, or GOI, and therefore provide a more obtainable application timeframe for the control of the tetrahedron structures.

The implemented procedure for getting robotic vehicles to operate and achieve autonomy has been accomplished by breaking down the whole into smaller parts and controlling each part each step of the way. This meticulous control scheme has commonly been referred to as giving too much intelligence to the system. How much of our expertise can we pour into a controlled system? The objective of researching and applying natural intelligence is that the intelligence will emerge to the situation and a meticulous control scheme will not be needed. The natural systems in our world and universe show signs of an emergent control or order and even self-organization [11], [12]. The application of those properties and others found in CAS can bring an unfolding of a 'natural intelligence', not necessarily similar to humankind but one of system oriented and goal oriented objectives [13]. Complex systems have emergent properties that can be beneficial to this research and the complexity of a framework for achieving the collaboration [14], [15], [16].

2 Background Research

Research and achievements in autonomous operation of complex systems, like robotics and UAVs, has been ongoing for many years. Government organizations, universities and industry have ongoing research and projects to develop state of the art autonomous vehicles [17]. Federal and military institutions contribute a wide variety of research dollars and future dependencies towards autonomous operation and navigation [18], [17]. The United States Congress has funded the military's UAV projects out to 2015 covering their semi-autonomous UAV's. Bone and Bolkcom presented the military UAVs in use today and outlined the major UAVs in development for future deployment [18]. The control methodologies applied to walking robots and robotics to date has resulted in small advancements in capabilities [4]. A specific task implemented in a controlled environment or system has fully been accomplished [6].

Each application of agents to any system requires a detailed understanding of the system components and interactions with other systems components and the environment in which the system must exist. The typical reductionism method of science does not apply with complex systems [12].

In order to address the problem of autonomy and multi-agent communication and control, studies from many disciplines will be investigated. There is a rich collection of prior research considered important to this research. Each application of agents to any system requires a detailed understanding of the system components and interactions with other systems components and the environment in which the system must exist. The typical reductionism method of science does not apply with complex systems [12].

The development of a valid framework for application to complex systems needs to include research in the area of autonomy beyond the typical expert systems and control methods. Holland discussed how the complexity of CAS contributed to the difficulty of understanding these systems and especially understanding how properties can emerge from these systems [12]. He outlined how simple laws could be studied that instead brought about the emergence of traits or properties that were of desirable outcome. The concept Holland discussed of something more evolving from simple laws or rules will be the driving goal of this literature research. Systems, architectures, algorithms, robotic frameworks, and research that show evidence of CAS properties emerging are of interest in the development of a framework or approach that will allow autonomy in multi-agents to emerge.

Holland discussed his understanding of complex adaptive systems and the hidden order in CAS as a separation of CAS into seven basic elements that includes three mechanisms, which he discloses as primitives, and four properties [11]. These mechanisms and properties allow the use of agents in building the model. The three mechanisms are tags, building blocks, and internal models. The four properties are aggregation, flows, diversity, and non-linearity. The key concepts from Holland are the interactions of the agents with each other and their environment. This appears to be a new way of looking at CAS. The important point drawn from this work is the fact that the agents can be set up to interact with the environment and have other properties emerge as a result.

2.1 Intelligent Systems (IS)

The reality of an intelligent system simulating human capabilities is still just a dream but there is ongoing research taking place. The American Institute of Aeronautics and Astronautics (AIAA) recently hosted a technical conference on IS to bring together research from the field of study [19]. The AIAA noted the benefit of the application of biological and cognitive systems for the efficient solution of complex problems. They also note the increase of IS technology applications in space and aeronautics. They realized the complexity involved with IS and a need to understand more clearly the logic, process, and information required to make systems safe efficient and productive. The conception, discovery, and reasoning with knowledge and information that IS technologies collect, evaluate, organize, and communicate were noted as important criteria to push the human-machine envelope of capability.

The objective of the AIAA technical conference in IS was disclosed as the need to enable an understanding of the history, concepts, and benefits from the applications of IS to aeronautics and space. Intelligent systems are required to advance the exploration of space and this advancement requires the application of new concepts in IS to provide safe and efficient space exploration [1].

Curtis et al. are looking into future space exploration [1]. The intelligence in systems and the interactions of these systems with astronauts needs development beyond our current understanding and capabilities. Their work in pushing the envelope of technology to expand exploration capabilities has been a key research in the NASA space initiatives for the past 5 years.

2.2 Complex Adaptive Systems (CAS)

Many researchers and authors have written books that explore the areas of complexity and complex systems and the benefits of what has been learned and applied [11], [14], [15], [16]. These benefits can have a direct application to research in the area of autonomy and multi-agent complex systems [12], [13], [20], [21]. There are many algorithms that have been developed drawn from natural complex systems, which could also have useful application to this research [22], [23].

Flake takes CAS and provides simple explanations and examples [20]. Flake looks at simplicity as the selection of sub elements or systems that can be compressed. Complex systems would therefore be complex and not simple or compressible. Flake covered complex systems, fractals, chaos, complexity and simplicity. Of particular interest in his work is the chapter on adaptation based in biological systems. Similar to many research papers presented in this paper, the idea that experience with the environment through acquired traits and learning is becoming a requirement for the adaptation of systems.

2.3 Artificial Intelligence (AI)

The techniques used in applications of AI towards development in autonomous vehicles have been advancing rapidly by including hybrid combinations of conventional engineering disciplines and AI techniques [6], [4], [8]. The consideration of Artificial Intelligence and CAS will be closely related for the scope of this research. Other emergent systems, such as Cellular Automata (CA), are currently available to research

and apply towards emerging needed qualities [24], [25]. CA represents a sort of simple intelligent rule system that evolves the rules into what appears to be intelligent actions or at least an intended outcome.

2.4 Intelligent Agents (IA)

The application of agent software has strived to represent humankind to some degree of intelligence. This representation has grown from simple task completion requests to combinations of intelligent autonomous task completion. Wooldridge discusses the rationality of agents operating under a Belief-Desire-Intention Model [25]. The author clarifies and bounds autonomy simply as the agent operating independently. The strengths in Wooldridge's arguments are advancement of communication and cooperation abilities among the agents. The problem solving capability is similar to a team of individuals working cooperatively on difficult problems to obtain a solution that no one individual could solve. This approach represents a logical application of additional resources to overcome stagnate problem solutions or dead-end thinking.

Progressing agents towards multi-agent systems increases the complexity of the systems. The objective of getting the desired outcome or the obtainment of a specific goal becomes major challenges with multi-agent systems. The cooperation and communication discussed by Wooldridge and others become important variables in the formula for successful completion of objectives.

3 Research Goals

The goal of this research is to develop a framework and approach to achieve collaborative autonomy in multi-agent systems and demonstrate its utility by applying the framework to a complex system – TETWalker. The application of this framework should allow the agents to achieve a level of autonomy. A single complex system, the TETwalker, will be chosen as the test platform and the framework will be applied to the system after the components are divided into multi-agents.

Collaborative autonomy is defined as the cooperative communication and work accomplished by the individual agents obtaining a collective goal. Each individual agent will have specific individual goals to accomplish and also participate in a higher goal that would require collaboration among the multiple individual agents. The autonomous actions of the collective individual agents will need to obtain the higher order autonomy required by the complex system. The collaboration required to control the TETwalker and walk the system similar to a robotic vehicle is not understood at this time and requires investigation.

Multi-agent systems consist of individual systems that have the multi-agent framework distributed within the system by multiple agents. With the achievement of this level of application, the framework can be expanded to multiple systems. The level of autonomy that will be achievable is not known at this time.

This research and framework can be measured by directly applying the framework to a simulation either of complex systems or through a simulation of multiple robots or multi-agents. Given a simulated robot such as the TETwalker, can the robot walk once the framework of multi-agent communication is applied? In essence, given a

complex system, can a small set of laws or rules be applied to achieve a collaborative approach to autonomy? The complex nature of applying a multi-agent approach is not known at this time. If a simulation proves the framework is successful, the research may stop at that point with the collection and analysis of the data under several repeated test setups. If physical robot systems are available and an application can be made, this approach may also be used to measure the benefits but this research does not depend on the physical application.

This research will also investigate the use and applicability of merging some of the properties of (CAS) such as emersion, adaptation, self-organization, evolution, autonomous agents, learning, and Artificial Intelligence (AI). The security of the agent communication will also be addressed during this research to help understand the consequences of disrupted communication.

The issues that face researchers and designers today of the applications of CAS techniques will be encountered and an understanding of those issues will evolve. Through this research, it is hoped that the applicability of merging several of these features and knowledge management techniques into a hybrid system can bring some understanding from the collaborative autonomy witnessed in natural multi-agent systems.

3.1 Tasks and Objectives

The current objectives are outlined as a list of tasks.

- Continue exploration of current research in CAS and Multi-Agent applications
- Develop the TETwalker model within a Graphical User Interface (GUI) that allows the retrieval of information from the physical model.
- Develop the model into an object that can interact within a 3D virtual world simulated on a computer. Exploration of 3D graphics programs may provide enough resolution to successfully operate the TETwalker model.
- Build a simulation where the interactions between the simulated world model and the TETwalker model provide a realistic account of collision detection, physics properties (gravity, friction, force, etc.) and time.
- Design the software and agent algorithms in a way to allow the inclusion of several CAS techniques. This will allow trying new CAS algorithms among the agents to develop the multi-agent communication framework.
- Implement the current design objective of robotic vehicle awareness and identification with the dynamic environment. The objective of providing a virtual point in the 3D space relative to the vehicle will be sought. This methodology should provide an attraction point for the agent algorithms to converge towards or diverge from in the simulation. Degrees of Membership with other relationships will also be added during this implementation.
- Finalize the multi-agent communication framework design that can provide a broad application to wide problem domains. Ease of use, simplicity, and the emergence of CAS properties such as collaboration, adaptation, and progression towards goal obtainment. These are several of the desirable properties. Others include adaptability and expandability that include the expansion from the four node TET model towards the twelve node TET model and beyond.

- Run simulation scenarios on the algorithms to test the walking or exploration gaits provided by manipulation of the agent behavior, communication, and goal obtainment.
- Record and plot data on the effectiveness of several gaits and simulated explorations.
- Publish the results in the final dissertation report.

3.2 Preliminary System Analysis

The current TETwalker system can be analyzed from many different perspectives. Three perspectives have been considered at the current research stage. Figure 5 graphically identifies the three agent divisions considered for application to the TETwalker system. The first approach (a) is to assign Agents to the nodes. This is the approach that will be studied by Walt Truszkowski and his student team at the Goddard Space Flight Center. This approach would only require four Agents and appears to be an optimal approach. A second approach (b) is to assign Agents to the struts of the system. This approach may be addressed latter but would need six Agents to implement. Another approach is to assign the Agents to represent a facet of the tetrahedrons as seen in figure 5 (c). This will be the approach pursued in this investigation. The facets represent a good component of the complex system. The facet is composed of smaller components and a communication framework can be shared naturally between the facets because each facet shares components with neighboring facets. Another benefit will be the representation of a facet in future multiple tetrahedron structures as either external shell members (similar to the exterior skeleton of

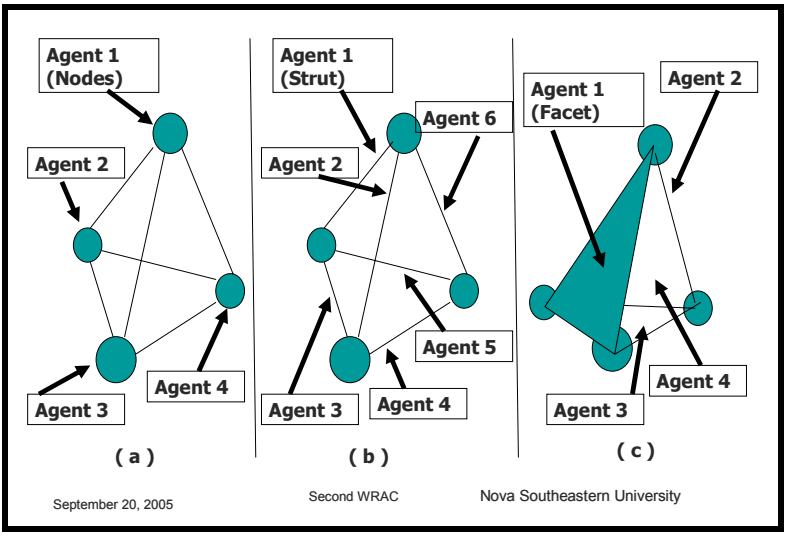


Fig. 5. Three perspectives for Agent representation for TETwalker component membership

an ant) or internal muscles and tendons required for supportive functions of morphing shape architectures. It is felt that this approach keeps more in line with the concepts of CAS and would allow a higher order of component representation and future expandability.

Collaborative autonomy would then depend on the sharing of data, information, knowledge, and goals. Each facet would still include a degree of membership allocated at the nodes. The Agents represented as facets should only require placement of the facet to a specified location. This would also allow each Agent facet to contain numerical attributes such as strut length, angle measurements, relative positions based on strut lengths and vector angles, and identity in relation to at least the system and possibly the environment through reference to the "spot" or reference point in the 3D virtual space that is simulated. This relationship among the facets and its members will be important as the TETwalker system is expanded to multiple systems.

It is proposed that the autonomous actions of the collective individual agents can then obtain the higher order of autonomy required by the complex system. The collaboration required to control the TETwalker and walk the system similar to a robotic vehicle may then be accomplished using known reflex actions or autonomic actions. These reflex actions would be moves or maneuvers that have already been accomplished or learned in the system.

4 Conclusion

CAS provides a unique non-reductionism approach to the advancement of research on complex systems like the NASA TETwalker represented in Dr. Steven Curtis's team research efforts [1]. Approaching this unique problem from a different approach as presented in this research effort can provide insight into the benefits of CAS techniques applied to complex systems. Another approach, as discussed with the project team, would be to provide simulated intelligence using a complex array of multiple parallel processors. This approach is logical but difficult to accomplish given the current state of processing horsepower and AI techniques. CAS provides a small window for scientific viewing and studying of the natural systems represented in nature today. Study in this area can bring understanding to imitating the autonomy, adaptation, collaboration, and emersion properties observed in natural systems [27], [28], [29]. The Agent approach and framework may even add to grouping future control schemes into manageable simpler processes that can cover distributed autonomy in complex systems.

It is hoped that the autonomous actions of the collective individual agents can obtain a higher order of autonomy and uncover actions that can be simplified as reflex actions or autonomic actions similar to biological organisms. A goal of this research dissertation is to contribute to autonomy through a small step in understanding complex systems such as the TETwalker. This research is striving to make a small contribution to autonomy and autonomic processes. The consultation of Goddard's agent expertise, through Walt Truszkowski, will bring beneficial contributions and direction to this research effort.

References

1. Curtis, S. A., Truszkowski, W. F., Rilee, M. L., & Clark, P. E.: Ants for the Human Exploration and Development of Space. Paper presented at the IEEE Aeorspace Conference, Big Sky, MT., 8-15 March. (2003)
2. Smith, S.: Open Dynamics Engine. ODE physics simulation software development kit http://www.itee.uq.edu.au/~wyeth/Publications/helicopter.PDF assessed (2004)
3. Chandler, P.R., Pachter, M., Swaroop, D., Fowler, J.A., Howlett, J.K., & Rasmussen, S. et al.: Complexity in UAV Cooperative Control. In Proceedings of the American Control Conference, (2002) 1831-1836.
4. Asama, H., Yano, M., Tsuchiya, K., Ito, K., Yuasa, H., Ota, J., et al. System Principle on Emergence of Mobiligence and Engineering Realization. Intelligent Robots and Systems, IEEE International Conference. October 2003, (2), (2003) 1715 – 1720.
5. Chandler, P.R., Pachter, M., & Rasmussen, S.: UAV Cooperative Control. In Proceedings of the American Control Conference, (2001) 50-55.
6. Wyeth G., Buskey G. & Roberts J.: Flight control using an artificial neural network. In Proceedings of the Australian Conference on Robotics and Automation (ACRA 2000), August 30 - September 1, Melbourne (2000)
7. Macdorman, K.F., Tatani, K., Miyazake, Y., & Koeda, M. : Proto-symbol emergence. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems, Takamatsu, Japan, November 2000, (3), (2000) 1619-1625.
8. Dufrene, W. R. : Application of artificial intelligence techniques in uninhabited aerial vehicle flight. The 22nd Digital Avionics Systems Conference, 2003. DASC '03, Indianapolis, IN, USA, October 2003, (2), (2003), 8_C_3_1- 8_C_3_6
9. Robinson, R. : Better than Human Flight Control Systems. http://gtresearchnews.gatech.edu/reshor/rh-win00/flight.html. Accessed May 10, 2003. Author's email: rick.robinson@gtri.gatech.edu (2000)
10. Page, D. : MAV Flight Control: Realities and Challenges. High Technology Careers Magazine, (1998)
11. Holland, J. H.: Hidden order: How adaptation builds complexity. Boulder, CO: Helix Books and Perseus Books (1995)
12. Holland, J. H. : Emergence: From chaos to order. Boulder, CO: Perseus Books (1998)
13. Bar-Yam, Y. : The Dynamics of Complex Systems. Perseus Books, Boulder, CO, 1 – 110, (1997)
14. Gell-Mann, M.: The Quark and the Jaguar: Adventures in the Simple and the Complex. W.H. Freeman and Company, New York, NY (1994)
15. Waldrop, M. M. : Complexity: the Emerging Science at the Edge of Order and Chaos. Simon & Schuster, New York, NY (1992)
16. Lewin, R. : Complexity: Life at the Edge of Chaos. The University of Chicago Press, Chicago, IL. (1999)
17. Bone, E., & Bolkcom, C.: Unmanned Aerial Vehicles: Background and Issues for Congress. http://www.fas.org/irp/crs/RL31872.pdf . Accessed September 30, 2003 (2003)
18. Glade, D. : Unmanned Aerial Vehicles: Implications for Military Operations (Occasional Paper No. 16, Center for Strategy and Technology, Air War College, pp. 17-19). Maxwell Air Force Base, CA: Air University (2000)
19. AIAA 1st intelligent systems technical conference. : Retrieved May 19, 2005, from http://www.aiaa.org/content.cfm?pageid=230&lumeetingid=1007&viewcon=overview#zz1009 (2004)

20. Flake, G. W. : The computational beauty of nature: Computer explorations of fractals, chaos, complex systems, and adaptation. Cambridge, MA: The MIT Press (1998)
21. Gleick, J.: Chaos: Making a New Science. Penguin Books, New York, NY, (1987) 1-32.
22. Cannady, J. : DCIS 790 Complex Adaptive Systems Slides. Class Lecture/Slides, from Nova Southeastern University, Graduate School of Information Sciences Web site: http://scis.nova.edu/~cannady . Lecture given March 5, 2004 (2004)
23. Bonabeau, E., Dorigo, M., & Theraulaz, G. : Swarm Intelligence: From Natural to Artificial Intelligence. Oxford University Press, Oxford, NY, (1999) 9–108.
24. Wolfram S. : A New Kind of Science. Wolfram Media, Inc., Champaign, IL., (2002)
25. Sarkar, P. : A brief history of cellular automata. ACM Computing Surveys (CSUR), (32), (2000) 80 - 107
26. Wooldridge, M.: Reasoning about Rational Agents. The MIT Press, Cambridge, Massachusetts (2000)
27. J. Buchli and C.C. Santini.: Complexity engineering: Harnessing emergent phenomena as opportunities for engineering. In Reports of the Santa Fe Institute's Complex Systems Summer School 2005. Santa Fe Institute, (2005)
28. A.J. Ijspeert, J. Buchli, A. Crespi, L. Righetti, and Y. Bourquin. : Institute presentation: Biologically inspired robotics group at epfl. International Journal of Advanced Robotics Systems, 2(2):175-199, (2005)
29. R. Moeckel, C. Jaquier, K. Drapel, A. Upegui, and A. Ijspeert.: YaMoR and bluemove - an autonomous modular robot with bluetooth interface for exploring adaptive locomotion. In Proceedings CLAWAR 2005, (2005)

Autonomy Without Independence: Animal Training as a Model for Robot Design

David C. Wyland

Reasonable Machines
165 Berkshire Drive
Morgan Hill, CA 95037 USA
dcwyland@ix.netcom.com

Abstract. A classic autonomous robot is an autonomous agent for open, unpredictable environments. Such an agent is inherently autonomous but not independent. Independence implies unpredictability, which is incompatible with agency. The current robot models – behavior based and artificial intelligence – have not been effective at implementing the classic autonomous robot model due to limitations in their definitions. The artificial intelligence model cannot deal with unpredictable environments, and neither model directly includes the concept of agency. Animal training as a model for robotics has the potential to avoid these problems. Animal training has several advantages. It has an inherent model of agency. Goals and behavior are formally separated into human goals and animal behavior. The animal is autonomous, requiring conversation between human and animal, but it is not an independent entity. A robot designed using this model is an articulate machine, programmed as an agent for the user.

1 Introduction

Animals can be trained to do amazing things. Examples abound in the form of the sheep dog, the cutting horse, the rescue dog, the working elephant and the seeing-eye dog, to say nothing of circus animals. Animals have been trained for various complex human support tasks for millennia.

The purpose of animal training, such as dog training, is to cause the animal to exhibit behaviors on command that will achieve the goals of the trainer. The trainer does this by enabling and modifying existing behaviors to create new ones.

When properly trained, a sheep dog will reliably do the sheep herding tasks given by its master, the shepherd. While working, the sheep dog acts as an agent for the shepherd, not as an independent animal. If a rabbit runs across the sheep dog's path, the dog continues herding instead of chasing the rabbit. This model of interaction between human and animal has some interesting characteristics.

1. The dog is trained as an agent for the human. The human has the goals in the relationship; the dog does not.
2. The human does the task design, selecting and modifying behaviors to achieve the desired goals.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 160 – 171, 2006.

3. The dog is not required to be intelligent nor conscious to be useful to us.
4. Tasks and their goals are formally separated from behavior.
5. The dog works in an open environment.
6. The behavior of the dog is probabilistic. It may not achieve a task on first try, and it will achieve it to a degree that can vary from one time to the next.

Let us consider this as a model for a robot, where the robot takes the place of the dog As in the above examples, the human is the trainer. We will assume a Behavior Based (BB) robot with the added ability to select and modify the behavior in real time by communication with the robot. If we implement the model correctly, it will have the following characteristics:

1. The robot is designed as an agent for the human. The human has the goals; the robot does not.
2. The human does the task design, selecting and modifying behaviors to achieve the desired goals.
3. The robot is not required to be intelligent nor conscious.
4. Tasks and their goals are formally separated from behavior.
5. Because it is behavior based, the robot works in an open environment.
6. The behavior of the robot is probabilistic. Because it works in an open, unpredictable environment, it may not achieve a task on first try, and it will achieve it to a degree that can vary from one time to the next.

The Animal Training (AT) model provides autonomy without independence. The robot acts autonomously in open, unpredictable environments, but its actions are completely dependent upon human design and direction. Rather than having an independent intelligence, we have a perfect dependent agent, one with no drives other than those we define and enable. This paper compares the AT model for robot design against previous design models in terms of their usefulness for designing a classic autonomous robot.

1.1 The Robot as an Agent for Open Environments

We can define an autonomous mobile robot as an agent for open environments, as discussed by Wyland in [1]. The robot as an agent for humans is inherent in our concept of the robot. We want it to do work for us. An open environment is unpredictable but familiar, and knowledge of the environment is inherently and chronically incomplete. It is unpredictable because events can happen spontaneously, such as a human or moving object interacting with the robot. The robot must operate on the basis of familiarity because it is never in exactly the same place twice and it never perceives its environment in exactly the same way twice. A robot working in an unpredictable environment implies autonomy. It must deal with its environment as it perceives it.

1.2 Agency Versus Independence

To an observer, an autonomous robot appears to “think for itself” as an independent entity as it reacts to its unpredictable environment. However, if we know its internal programming well enough to accurately predict how it will react to its environment,

we no longer consider the robot as thinking for itself, but consider it as a predictable automatic machine. Therefore, a robot or other entity (animal, human, etc.) is independent to the degree that we cannot accurately predict how it will react to its environment.

We want the robot to be predictable in all environments. Unpredictable behavior in any environment could be dangerous. Properly designed, the robot will act in a predictable manner for a given task in a familiar environment. We also want it to act in a predictable manner in an unfamiliar environment. If the robot is in an unfamiliar environment, we want it to have a well-defined, default behavior, such as stopping and asking for help.

Robots can be unpredictable in unfamiliar environments if not designed for them. To be predictable in an unfamiliar environment, the robot must detect when the current environment is sufficiently different from the environments it knows and respond with a default behavior. Many machine-learning mechanisms, such as neural networks and genetic algorithms, do not inherently have such a novelty detection mechanism and are vulnerable to unpredictable behavior. Even simple task and behavior designs need to verify that the current environment is suitable for current task to remain predictable.

The concepts of agency and independence are incompatible. Agency means that the agent works for the user, and the device makes predictable decisions defined by the user. Independence means that the independent device makes unpredictable decisions on its own. An entity can be an agent to the degree that it is predictable; it is independent to the degree that it is unpredictable. We can make an independent entity an agent only to the degree that we make it predictable, i.e. to the degree that we eliminate its independence.

An agent needs to be autonomous but not independent. Autonomy and independence are different concepts. Independence implies autonomy, but autonomy does not imply independence. To be independent, you must be autonomous, or the concept has little meaning. However, you can have autonomy without independence. A thermostat is an example of autonomy without independence. It autonomously decides when to turn the heater on and off, but its user sets its temperature goal. It serves as an agent for the user. A sheep dog has autonomy but not independence. It does the tasks the shepherd assigns.

We want our robots to be autonomous but dependent machines because we want them to be safe and reliable. This means that for a given environment and task, the robot's actions are completely predictable.

2 Robot Design Models

A design model captures the problem to be solved by the design and the approach to solving it. To select a model, we need to identify the requirements of the design. In our discussion of robots, we defined them as agents for open environments.

The first requirement is to be able to work in an open environment. An open environment is unpredictable but familiar. Homes, offices and the outdoors are examples of open environments. These are open because people and animals come

and go, and furniture is rearranged. They are also open because the robot will never be in exactly the same place twice and will never see its environment from exactly the same pose twice.

The second requirement is agency. Agency is the ability to carry out useful tasks for others, e.g. humans. As we require this ability in a robot, we need to be specific about the qualities of agency we require. We want to be able to give the robot a task at any time and have it respond in a timely manner, where timely is measured against human performance. If adding a new task requires days or weeks of design and implementation time, the robot does not meet our requirement of timely response.

2.1 Current Design Models

There have been two major models in robot design: the artificial intelligence (AI) model and the Behavior-Based (BB) model. The AI model is concerned with modeling human intelligence, as its name implies. The image of the completed AI model is an android, a synthetic human. The BB model is derived from the cybernetic tradition as defined by Weiner in [2], which is concerned with communication and control in human and animal behavior. Its image is a synthetic animal or insect.

2.2 Artificial Intelligence (AI) Model

Artificial Intelligence (AI) as a formal discipline developed a particular model for robot design called the Sense-Model-Plan-Act, or SMPA model, as discussed by Brooks in [3]. You sense (i.e., measure) the environment, incorporate the measurements into a model of your world, create a plan to execute a given task by analyzing that model, and carry out the actions defined by the plan. If nothing in the environment changes, you go through this sequence once to execute a task. If anything changes during plan execution, you repeat the SMPA cycle as required.

A key concept in AI is the planner as an automatic task designer. The planner designs the target task by using mathematical logic to select the best task from a suite of reasonable or possible task designs. Some of the first AI planners such as the General Problem Solver described by Newell, Shaw and Simon in [5] were based in part on theorem proving programs, such as the Logic Theorist also by Newell, Shaw and Simon in [6]. In this view, designing a plan is reduced to an algorithmic process, like solving an equation. For a discussion of AI planning and its problems, see Brooks [7] and Pollock [8]

In practice, AI planners are less automatic than intended. As Pollock [8] points out in reference to the STRIPS planner developed by Fikes and Nilsson in [9], "… What is important here is that the planning is interleaved with epistemic reasoning aimed at finding appropriate information for use in the planning, and the course of the epistemic reasoning is directed by where the agent is in its planning. In effect, requiring the human operator of an AI planner to perform this epistemic reasoning in advance requires him or her to already know how the planning problem will be solved, but this makes the planner superfluous." In this view, AI planners look more like pre-designed plan repositories than independent plan designers.

The AI model depends on a model of the world and the robot within it. Given complete knowledge of the world the robot inhabits, such a model can be manipulated

to generate plans for tasks. Industrial robots in factories occupy this kind of closed, known world, and their performance has been good as a result. This model also works well in spacecraft robots. All reasonable failures and failure combinations can be programmed into the internal model of the craft, and an action is defined for each failure combination. All spacecraft states are therefore known, and the appropriate action is defined for each state.

However, the AI model does not work well in open environments, where knowledge of the world is inherently incomplete. AI uses logical models of the world to predict the future from the state of the current model and use these predictions to form a plan. In an unpredictable world, this is either logically impossible or computationally intractable. A variant of this problem is the famous frame problem, discussed by Ford and Pylyshyn in [4] which is the problem of predicting what does not change in the world when the robot changes some thing. For a discussion of other problems with the AI approach, see Pollock [8], Ford and Pylyshyn [4] and Brooks [3]. The AI approach does not work in open environments, so it fails our first requirement.

The AI approach also does not have an inherent concept of agency. AI implies independence. As a synthetic human, an artificially intelligent machine should independently think for itself. There is no direct path to the inclusion of agency. The tacit assumption seems to be that once we learn how intelligence works, we will be able to design a robot that will naturally act as an agent for us. Incidentally, the concept of planning and using planners supplies an indirect tradition of agency. Planners require goals as inputs to the planning. We do not understand AI well enough to model it as a useful generator of these goals, so human users typically supply them. The robot is acting as an agent, but only temporarily to support development.

2.3 Behavior-Based (BB) Model

The Behavior-Based (BB) model focuses on creating desired behavior in a robot. Arkin in [10] defines behavior as: “A behavior is (…) a response to a stimulus.” Robot behavior in an environment is a response to stimulus from the environment as perceived by the robot. This can be an outside stimulus such as a light or sound, or it can be a stimulus caused by the robot sensing an obstacle as the robot approaches it. The BB model is a mapping of stimuli to responses. It has evolved along with the AI model, with both having origins at around 1950. The BB model is associated with cybernetics, the study of communication and control in man and machine, as originally defined by Weiner in [2].

BB robots can be divided into two categories. The first category is reactive systems and contains pure stimulus-response BB robots that have neither state nor planning. The second category is hybrid systems and adds state and planning elements to the reactive systems model.

Perhaps the first example of a BB robot was the W. Grey Walter’s mechanical tortoise described by Walter in [11]. It looked a little like a tortoise and moved around the floor on three wheels, a single wheel for driving and steering, and two idling wheels. It had two sensors (a photocell and a bump sensor) and two actuators in the form of motors, one for the drive wheel and one for steering. It was a simple robot

with three behaviors, but it demonstrated what appeared to be sophisticated, even goal oriented activity. Simple behaviors and a complex environment produced complex activity.

In reactive systems BB robots, behaviors are designed in and selected at run time to produce the desired robot activity. Artificial neural networks have been used to allow BB robot behaviors to be trained-in rather than designed-in, but the intent is the same. Such robots resemble mechanical animals or insects, particularly those with multiple legs. Indeed, animals and insects are studied for ideas in behavior that can be applied to BB robots. This is called ethologically guided or constrained design. See Brooks [7] and Pollock [8] for a more thorough discussion of BB robotics.

However, the reactive systems BB design model does not have the problem with open environments that the AI model has. It reacts directly to the environment as it experiences it, whether open or closed. It has no model of the environment to be incomplete or in error. It is not bothered by unpredictability because it makes no predictions. This meets our first requirement of being able to operate in an open environment.

The reactive systems BB model does not include agency, as we have defined it. As a synthetic animal or insect, it is expected to think for itself. It maps stimuli to responses with no memory other than the mapping itself. Since it has no state memory, there is no way to give it a task to remember or ask it about tasks completed. Current floor-cleaning and lawn-mowing robots are of this type. However as Brooks in [3] points out, the BB model can certainly have state and is not constrained to be purely reactive. For example, the Allen robot described by Brooks in [12] had a higher layer that would select a goal for the lower, reactive layer to achieve.

If the reactive systems BB model has no state memory, it cannot include our concept of agency, where agency is considered to be a real-time request for task performance.

2.4 Hybrid Model

Hybrid models are combinations of the deliberative elements of AI and the reactive elements of BB in an attempt to use the advantages of one model to compensate for the deficiencies of the other. Many combinations are possible, from primarily AI to primarily BB. Arkin in [10] provides a review of many of these hybrids.

Combining the AI and BB models is problematic. The AI model relies on knowledge contained in a model in order to plan and generate actions, while the BB model does not have such a model. The AI model retains all the problems of rational planning, and there is no obvious way for the BB model to help this. A major problem of hybrid models is how to combine deliberative and reactive elements in a working design. As Arkin in [10] notes, “The nature and the boundary between deliberative and reactive execution is not well understood at this time, leading to somewhat arbitrary architectural decisions.” Also, planning that uses behaviors will be much different than conventional AI planning, as pointed out by Brooks in [7] and by Agre and Chapman in [13].

The intent of the hybrid model is to add planning to the reactive BB model. The goal is to provide task planning capable of operating in an open environment.

Assuming that the hybrid model is based on reactive behavior, it should be able to operate in open environments, meeting our first requirement.

Like the AI and the reactive systems BB models, the hybrid robots have no direct concept of agency. Both assume that the robot is an independent, synthetic animal or insect that thinks for itself, violating our requirement for agency.

3 The Animal Training (AT) Model

The Animal Training (AT) model provides us with an image and a design methodology for a robot that can potentially meet our design requirements. 1) Because it is behavior based, it will work in open environments. 2) The human designer/user does the task design, providing a direct model of agency.

3.1 Robot Design Using the Animal Training Model

In the AT model, the robot designer is in the position of the dog trainer, and the robot user is in the position of the shepherd. Both are involved in the robot design, but at different points. The robot designer creates a robot with the physical characteristics and behavior capabilities useful for its range of tasks in its set of environments. The robot user then designs task programs that use these primitive behaviors.

User programming of the robot is significant because a robot is an agent designed to do tasks for humans in open environments. All open environments are unique but with discoverable similarities. For example, even though two apartments may be identical in layout, the owners will furnish them uniquely. However, both apartments are similar because they will have sofas, chairs and tables, although different in type and style. As a result, each robot task design is unique to its intended open environment. Each user must design the tasks for the robot's unique environment.

Human task design for robots is finite in scope in the same sense that dog training is finite in scope. The robot is not expected to deal with all possible situations, just the typical ones. If it encounters a situation that it cannot deal with, it calls for help. An independent entity does not have this option; in principle, there is no one to call. As a result, human task design starts with a few practical tasks and the ability to deal with a few typical situations, such as avoiding an obstacle in the path to a goal. Assuming tasks can be added and extended incrementally, the robot can become subtle and sophisticated simply by accretion.

Robots in the AT model are not independent entities. They have no beliefs, desires or intentions of their own. Since robots in the AT model are not modeling independently acting humans or animals, there is no requirement for their design to be biologically plausible. However, animals and humans can be useful sources of design ideas.

3.2 Robots as Pick-and-Place Machines

Robots are designed to do tasks, and their tasks and environments largely determine their design. A house robot, an undersea robot and a planetary explorer robot are significantly different in their designs because of their very different tasks. However, they are all robots, so it is reasonable to ask what design characteristics they have in common.

As a broad generalization, robots are pick-and-place machines designed for open environments. Pick-and-place is a term used in industrial robotics to indicate that the robot picks up something from one location and puts it in another. Picking up includes manipulation of the object selected. Exploration robots are pick-and-place robots that seldom actually pick up something. Their job is to find interesting things, things that would be worthy of picking up. Manufacturing, cleaning and maintenance robots extend the definition of the picking-up to include manipulation of the things chosen for pick-up.

The combination of the expected task and its environment determines what is picked up and where it is placed. A house robot may get you a soda from the refrigerator, then pick up and put away the children's toys. A construction robot may unload wood from a truck and distribute to building sites. A planetary rover robot may explore an area looking for candidate objects to pick up, or to simply identify.

Since this is a broad generalization, there will be robots that do not seem to fit the model of pick-and-place. An example would be a social robot, such as a museum guide. However even in these cases, their implementation may be a subset of the pick-and-place model due to the need to work in open environments, for agency and for communication.

3.3 Pick-and Place-Requirements

Pick-and-place operations require several activities: 1) The robot must move to where the object for pick up is to be found, 2) It must search for and identify the object. 3) It must move to the object and pick it up, and 4) It must carry the object it to its destination and place it there. These operations require movement, navigation, object recognition, object grasping, pick-up, place and release.

A pick-and-place robot can be described as a robot arm with a platform to move it around and a vision system to guide it. The first requirement is a platform that can move around efficiently in the robot's intended environments. This is a challenging mechanical engineering design problem, one that can be simple in definition but complex in execution.

The robot needs to navigate relative to the objects in its environment because it needs to either go to them or avoid them. This kind of navigation is called pilotage, as in piloting a ship in a harbor. Pilotage is required for maneuvering in an open environment where objects can move around. Docking is precision pilotage, as in bringing the boat next to the dock. Docking is required for object manipulation, such as picking things up.

Robot pilotage requires recognizing objects (e.g. landmarks) and determining their location. It is also required for finding and picking up objects. Recognition of specific objects may not be required for collision avoidance, but is helpful even then. Because object recognition and location is central to our pick-and-place robots, their design can be called Object Based Design (OBD).

As an example of a pick-and-place house robot, consider a robot that will get you a soda from the refrigerator in the kitchen and bring it to you as you sit on the couch. To begin with, the robot must go to the kitchen. By recognizing surrounding landmarks, it can determine its location and select a path that will take it to the door into the kitchen and into the kitchen itself. Then, it will look for and recognize the

refrigerator as another landmark. The robot will proceed to the refrigerator and dock in front of it. Then, it will search for and recognize the door handle. The robot opens the refrigerator door, searches for a can of soda, recognizes and locates the can, guides the arm and gripper to the can and picks it up. The robot then closes the refrigerator door, navigates back to you and hands you the soda. As this shows, object recognition is essential for pick-and-place operations.

3.4 Object Recognition

Object recognition – particularly using vision - has been an area of intense study for decades. The results of these efforts have tended to be limited and specific rather than general. A recent advance offers to change this situation. The advance is a new vision based recognition algorithm, the Scale Invariant Feature Transform (SIFT) developed by David Lowe, as described in [14], [15] and [16]. This transform can uniquely recognize objects at any distance or pose and with varying lighting and occlusion. This capability is what the object recognition community has been seeking.

The SIFT algorithm can identify an object and measure its location in the image. Given knowledge of the camera optics and direction of gaze, the object bearing (angle to the robot) can be found. Using stereo vision or other distance measuring techniques, the range of the object can be found. The result is an identification of objects in the field of view and their location relative to the robot. Given algorithms like the SIFT algorithm that can provide rapid and reliable identification and location of all objects in view, pilotage, docking and grasping are straightforward engineering problems.

3.5 A Robot Architecture Using the Animal Training (AT) Model

The AT robot design model consists of a robot with useful behaviors and tasks programmed by a human user and that executes those tasks on verbal command. Physically, it is a mobile platform carrying a camera for object recognition and an arm for grasping and manipulating.

A robot architecture that can implement this model consists of a 1) a behavior manager, 2) a task manager and 3) a dialog manager. The behavior manager processes sensor data to drive actuators, such as the motors in the wheels (or legs) and arms of the robot. The task manager converts behaviors into tasks that achieve a goal, and it selects and executes tasks given to it by the dialog manager. The dialog manager is the interface between the human user and the task manager. It communicates with the task manager to select tasks and to create new tasks.

The behavior manager uses the identities and relative locations of objects to control the various actuators of the robot, as determined by the active behaviors. It consists of sensor processors (e.g. the vision system) to identify and locate objects, an analog logic system as described by Wyland in [1] to convert this information into actuator control signals, and actuator processors for local control of actuator motors.

The task manager executes tasks previously supplied by the human user. In this paper, we will assume the RAP model described by Firby in [17] for behavior based task management. Tasks consist of primitive tasks and composite tasks. A primitive task is a behavior that achieves a goal. The behavior is selected and enabled by the

task manger. The task manager also determines when the task is done and whether it was successful. A composite task is a tree structure of sub-tasks and primitive tasks.

Because the robot operates in an unpredictable environment, tasks can succeed or fail, and they can succeed in more than one way. If a task fails, the task manager must know what to do. An optional retry list for each task can serve this function. If a task fails, the task manager tries the next task on the retry list. If it reaches the end of the list without success, it passes the failure back to the task that called it or notifies the human at the appropriate time. This notification is a task of its own. Retry lists are often used by humans. For example, you may have a list of things to try if your car will not start.

The dialog manager provides structured natural language communication between the task manager and the human user. The robot only contains what we put in it. We enter the phrases and sentences we want it to recognize, and we enter the responses to those sentences. In concept, this is much like a frequently asked questions (FAQ) list. The difference is that the answers will reflect the current state of the robot, such as tasks completed. A simple conversation processor such as described by Suerth in [18] can be used as the dialog manager. Although the robot may eventually acquire a large number of input sentences and their responses, this is not as daunting as it may seem. The robot only needs to recognize the sentences that initiate its tasks, including replying to status questions. Sentences and their responses are added incrementally as needed for new tasks. Initially, only a small number of sentences and tasks may be required for the robot to be useful. This number can grow easily and incrementally with time. Eventually the robot may be capable of sophisticated tasks and conversation by simple accretion. In effect, the robot will contain the complete policy manual for a perfect clerk, where every useful, relevant question or request will be recorded, along with its appropriate response.

The initial dialog processor can be purely text based. This simplifies the recognition problem. Speech synthesis can be added for a verbal response, and speech recognition can be added for verbal input. Speech recognition – typically a difficult problem – can be aided by the finite nature of the vocabulary and dialog.

The dialog manager includes an episodic memory. This memory records task information from the task manager, such as tasks completed and objects recognized as part of task execution. The episodic memory remembers tasks executed and information specific to the task. Note that scanning for objects is a task. This limits and defines the scope of the memory. The episodic memory allows status reporting and history analysis, which is useful in task design and debug.

3.6 Task Design by Giving Directions

Robot design is about task design. In the AI and hybrid models, tasks are supposed to be designed automatically. In the BB model, tasks are designed-in as primitive behaviors. In the AT model, tasks are designed by the robot users.

The ordinary activity of giving directions is a form of task design, also known as task planning, as pointed out by Agre and Chapman in [13] and by Payton in [19]. Directions are commonly given for navigation purposes, as a task plan for going somewhere from a known starting point. Consider the task “Get me a soda.” Detailed directions for this task might be: 1) Go to the kitchen, 2) Go to the refrigerator,

3) Open the refrigerator door, 4) Find a soda, 5) Reach in and grasp the soda, 6) Pull the soda out of the refrigerator, 7) Close the refrigerator door, 8) Return to me (the requestor), 8) Present the soda to me, and 9) Release the soda can when I (the user) grasp it. These directions assume that each step (i.e. subtask) is a task that the human (or robot) knows how to do.

Directions are object based. Each step of these navigation directions refers to an object as the focus of the task. Most steps refer to movement from the current location toward a named object or a named place defined by objects around that place. Other steps involve manipulation of a target object, such as opening the refrigerator door and grasping the soda. The implied primitive tasks for these directions could be movement from the current position to a position near an object; grasping, opening and closing the refrigerator door; and grasping, picking up and releasing the soda can.

3.7 Task Memory and Task Selection

In the Animal Training (AT) model, the user selects previously designed tasks. This is planning by remembering, as described by Hammond in [20]. Tasks are named, and the dialog manager can select tasks to be done by name. Task design by giving directions implies each step of the directions is a subtask with a named object-based activity as its target. The task manager includes a task memory that holds the task designs.

Planning by remembering in the AT model does not mean the rejection of other forms of planning. In the process of creating tasks and using the robot, patterns of use and programming will develop. These patterns can be exploited to support portions of the design process.

3.8 Building the Robot – Work in Progress

We are building a robot based on the Animal Training design model as described in this paper. The first prototype is a battery powered, autonomous, indoor robot. It consists of a 12” square platform of a conventional design with two driven wheels plus a caster. The platform holds a small robot arm, a video camera and an embedded PC motherboard running Linux. It will be used to integrate the AT elements in a complete working robot.

4 Summary

The Animal Training (AT) design model allows us to design robots as articulate machines that do tasks as agents for human users in open environments. It provides an inherent model of agency, with a formal separation of human provided goals and robot behaviors. Neither consciousness nor intelligence is required or even meaningful because the robot is programmed as an autonomous machine, not an independent entity.

The AT model is behavior based, allowing the robot to function in an open, unpredictable environment. Its behaviors are designed-in, and the human user programs all its tasks. The task design problems of AI are avoided by relegating the designs to the human users. Object based behavior design and task design by giving

directions simplify these designs. Robot task sophistication grows naturally by accretion. The result is an an articulate, autonomous agent that performs tasks for humans in an open environment.

References

[1] David C. Wyland, Reasonable Machines: Analogical Reasoning in Autonomous Agent Design, Innovative Concepts for Agent-Based Systems, *First International Workshop on Radical Agent Concepts (WRAC 2002),* ed. Walt Truszkowski, Springer 2003, ISBN 3-540-40725-1.

[2] Norbert Weiner, Cybernetics, Second Edition, MIT Press, 1948.

[3] R.A. Brooks, Intelligence Without Representation, *Artificial Intelligence Journal (47)*, 1991.

[4] K. M. Ford and Z. W. Pylyshyn, The Robot's Dilemma Revisited, Ablex Publishing Company, 1996, ISBN 1-56750-142-7.

[5] Allen Newel, J. C. Shaw, Herbert Simon, " A General Problem Solving Program for a Computer," *Computers and Automation 8(7),* 1959, 10-16.

[6] Allen Newel, J. C. Shaw, Herbert Simon, "Empirical Explorations with the Logic Theory Machine," *Proc. Western Joint Computer conference, 15*, 1957, 218-329.

[7] R.A. Brooks, "Intelligence without Reason," Proceedings of the 1991 International Joint Conference on Artificial Intelligence, pp 569-595.

[8] John Pollock, "Planning Agents," ed. Rao and Wooldridge, Kluwer Academic Publishers 1999; ISBN: 0792356012

[9] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence 2*, pp 189-208.

[10] Ronald C. Arkin, Behavior Based Robotics, MIT Press, 1998, ISBN 0-262-01165-4.

[11] W. Grey Walter, "The Living Brain," W. W. Norton, New York, 1963

[12] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2, April, 1986 14-23.

[13] Philip Agre & David Chapman, "What are Plans For?," *Designing Autonomous Agents*, ed. Pattie Maes, MIT Press 1990, ISBN 0-262-63135-0.

[14] David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110.

[15] David G. Lowe, "Object Recognition from Local Scale-Invariant Features," *International Journal of Computer Vision*, Corfu, Greece (September 1999), pp. 1150-1157.

[16] Stephen Se, David Lowe and Jim Little, "Local and Global Localization for Mobile Robots Using Visual Landmarks," *IEEE International Conference on Intelligent Robots and Systems, IROS 2001*, Maui, Hawaii (October 2001), pp. 414-420.

[17] R. James Firby, "Adaptive Execution in Complex Dynamic Domains," Ph.D. Thesis, *Yale University Technical Report YALEU/CSD/RR #672,* 1994.

[18] Russell Suereth, Developing Natural Language Interfaces, McGraw-Hill 1997, ISBN 0-07-913017-8.

[19] David W. Payton, "Internalized Plans: A Representation for Action Resources," Designing Autonomous Agents, ed. Pattie Maes, MIT Press 1990, ISBN 0-262-63135-0.

[20] K. J. Hammond", "Case-Based Planning: A Framework for Planning from Experience", *Cognitive Science*, volume 14, pages 385--443, 1990

Shaping the Future of Online Payment Processing: An Autonomic Approach Applied to Intelligent Payment Brokers

Marcelo Perazolo, Carlos Hoyos, and Viswanath Srikanth

IBM Corporation
4205 S Miami Blvd, Durham, NC 27509
{mperazol,choyos,ahs}@us.ibm.com

Abstract. As e-Commerce systems evolve, and B2B models flourish, more and more complex payment transactions are performed electronically. A consistent topic periodically addressed by e-Commerce systems is how to reduce the costs associated with these electronic transactions, such that it is of minimal detriment to the growth in the volume of transactions that are conducted online. This question is of particular interest in the area of micro-payments, where the cost of the transaction is a significant percentage of the total cost of the goods exchanged between the merchant and the buyer. This paper proposes a novel architecture and associated artifacts for online payment processing providers and their client applications (online order capture and processing systems, payment capture and processing systems, and other online-based merchant software). New data structures, message exchange patterns and optimization algorithms are developed to support this new paradigm. These artifacts are molded following the concepts defined by the Autonomic Computing and Intelligent Agents disciplines.

1 Introduction

Large customers of payment service providers today do not have an easy way to automatically choose how to best process their online payment transactions, because payment providers have a static pricing structure and these prices are set as part of the service agreement between providers and merchants. Any changes in this pricing structure would mean that a new agreement would have to be drafted between the parties involved, making this architecture very unfriendly to dynamic cost variations inherent to a competitive landscape. This situation, unfortunately, does not benefit the end customers that ultimately pay for these costs since they are embedded as a percentage of the consumed goods and services. If customers of online payment processing systems would follow an autonomic approach based on mediation provided by intelligent payment brokers in order to dynamically and continuously evaluate how to best process a batch of electronic payment transactions, then competitiveness among payment providers would increase, resulting in a reduction in

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 172 – 183, 2006.

the cost overhead, a corresponding improvement in operating margins, and/or a reduction in the selling prices of goods and services offered to end consumers.

As examples of real world payment service providers and their respective payment protocols, please refer to [MC], [VISA], [SET], [ACH], [PAYPAL], [CYBER], [PARA], [VS], [FIRST] and [CITI]. There are many more, but this is a representative list of the institutions commonly available for online merchants. Additionally, a good reference for information on micro-payments is [MICRO].

Our proposed architecture consists of the following elements:

- An intelligent and autonomous payment processing broker
- A common interface between broker and merchants
- A common interface between broker and payment processors

The UML component diagram shown in Figure 1 depicts these elements and their relationships with other software elements in the merchant and provider side. For a reference on UML collaboration and other diagrams, please refer to [UML].

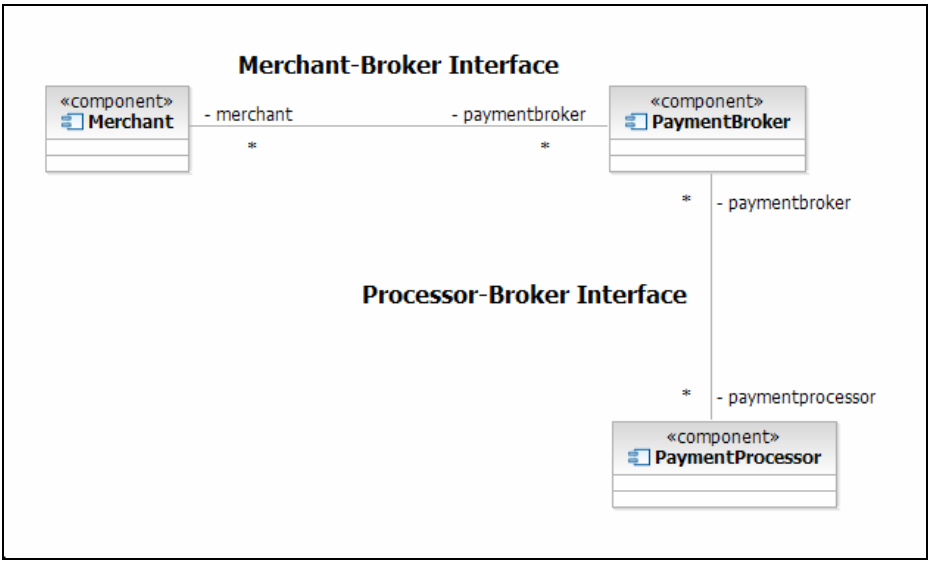


Fig. 1. Payment Broker and associated elements

As we can see from the diagram, the payment broker is a component that will mediate service requests between clients who need payment processing and payment processing service providers. This enables the mediator to provide real-time analysis of how to better process a given request for a batch of payment transactions. This mediator (the broker) has autonomic, adaptive and intelligent characteristics because it is constantly performing cost-benefit analysis based on previous history and current cost structures in order to determine how a request should be honored.

2 The Payment Broker Architecture

As we put together the architecture capable of supporting the concept of an intermediate payment broker, we must define what are the data structures and interfaces supported in the architecture. The following UML collaboration diagram further depicts how the components of this architecture communicate with each other in order to provide the mediation for payment request processing. The payment broker's role is to determine the best way to process a certain payment batch request using the captured payment data and the payment policies advertised by each payment processor. This information is then transmitted back to the merchant so it can select the best option, and it also augments the local knowledge base used to help in the analysis of future payment requests.

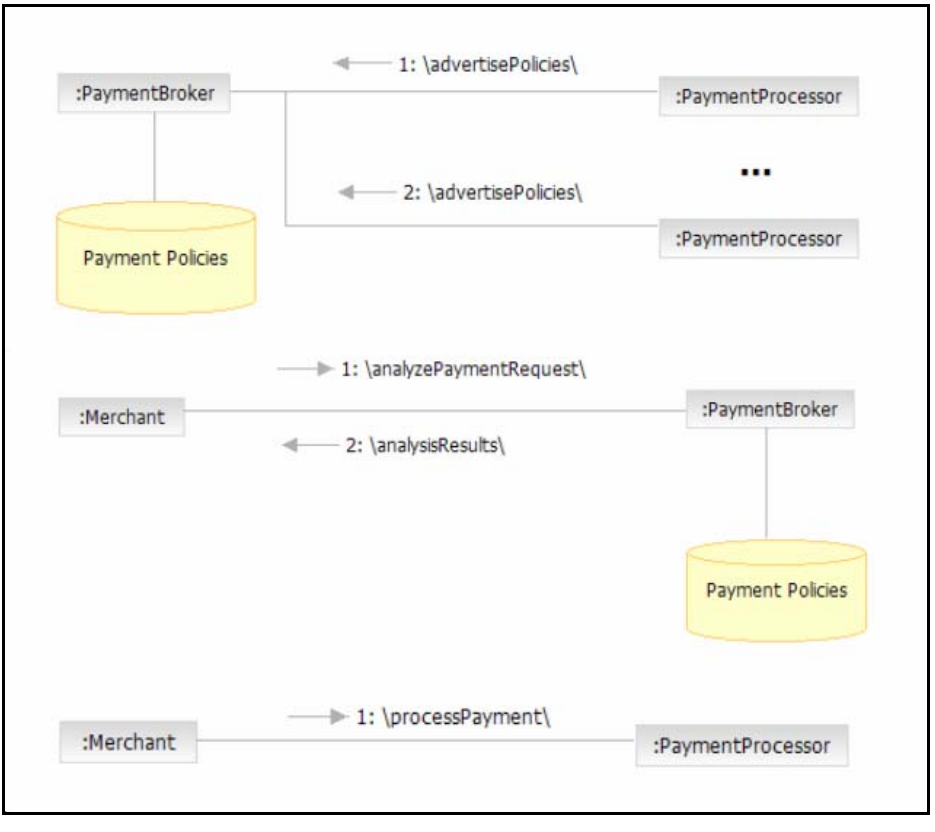


Fig. 2. 3-phase interaction model for payment brokerage

The previous figure shows a 3-phase interaction model that depicts the communication between the payment processor and the payment broker as well as the communication between the merchant application and the payment broker.

The processor-broker interface enables policies associated to each payment processor to be published and made available as parameters for the cost analysis algorithm present in the payment broker. This interaction is not frequent; it only needs to happen at setup time or when there are changes in the set of policies a certain payment provider supports (in which case, these changes must be communicated back to the payment broker).

Likewise, the merchant-broker interface enables payment instruction data and requests for cost analysis to flow between the merchant application and the payment broker. It also enables the results of the cost analysis to get back to the merchant application that can then effectively execute the requests. Alternatively, the analysis results of the payment requests and selection of the payment provider could be performed directly by the payment broker in behalf of the merchant application.

The cost analysis performed by the broker is said to be adaptive and intelligent because complex factors are often involved in the structure of payment transaction costs, like which are the products being purchased, what is the requested payment vehicle (payment method), time of request, contract restrictions, total number of transactions required by the merchant, total amount associated with each transaction, etc. All these variables are part of the information governed by the payment policies associated to each payment processor. Additionally, by adding a mediation layer between merchants and providers, other complex factors can be taken into account as well, so pricing structures can be better modeled and a higher degree of accuracy achieved.

In this paper we show typical factors and policies adopted by payment providers and how they can be factored in common metrics that are used as input for a dynamic and autonomous analysis of the minimum cost associated with any given payment processing request.

2.1 Payment Processor Policies

Several variables are typically accounted when a payment processor determines its pricing and operation model. Most of these variables can be translated into payment policies and are frequently disclosed as part of the agreement established between the payment processor and the merchant. Common policies frequently used in the payment processing industry are now presented:

Response Time: This policy determines the quality of service offered with each payment operation. Normally merchants submit thousands of different payment transactions to a provider. The contracted terms between the merchant and the payment processor must guarantee that this many transactions will be processed in a timely basis, otherwise merchants may experience higher and higher response times from payment processors and be limited in the number of transactions they can support in real time in their front end business. Typically large merchants contract higher response time, depending on the number of servers and transactions they support in their online store fronts.

Transaction Volume: This policy measures how much business will be consumed by a certain customer of a payment processor. The greater the transaction volume the more processing resources will be consumed, but at the same time the more profit it will bring to the payment processor. Payment processors can differentiate their level of service based on transaction volume expected by a certain merchant. Usually merchants with enough transaction volume justify higher quality of service and cheaper per-transaction rates. In addition, penalties could be established when a certain volume of transaction is not achieved during a certain subscription period.

Risk Factor: This policy tries to establish a measurable factor in the risk incurred by payment processor when it accepts a request for payment originated by a given merchant. At this point funds must be transferred between two different parties and the payment processor must be able to guarantee that transaction, accepting the business risk of payment default. Risk factor policies try to quantify the likelihood that merchants will be able to honor their debts with the payment processor at a later time. Usual aspects considered are: if the merchant has established credit in the market, if the merchant if an established business as opposed to a startup venture, and also the aspects of the merchant business itself - high risk businesses, like gambling, online adult services and non established newsletter services are usually rated poorly and have a higher associated risk.

Expiration: This policy determines if a payment transaction has an associated expiration time or not, and how long it takes for each payment transaction to expire. Usually this is used in conjunction with authorize operations, where the payment processor only reserves a certain amount in the account of the payer instead of actually committing it immediately. It is advantageous to have an expiration time associated to these reservations, because sometimes the merchant interaction (order, service, etc) doesn't go through all the way, and instead of issuing a new operation to cancel the previous authorization, a merchant could choose to just let it expire instead.

Batching Support: This policy determines if a payment provider supports batch processing of transactions in addition to real time processing. Sometimes it is more advantageous to a merchant to batch transactions and submit them all at once in periodic intervals or off peak hours rather than having to contract a very high response time with a payment processor. There are also limits to the real time processing capability of payment systems. When these limits are surpassed, it generally means that a batching capability is required to support the business model of a certain merchant.

Supported Validations: This policy determines if a validation mechanism is offered or not by the payment processor. It can also determine different levels of validation offered to merchants at different service levels. Depending on local laws in the payment processor geography a certain level of validation is required. As an example we mention the AVS (Address Verification System) required to payment providers offering credit card services in the United States.

Supported Operations: This policy determines the types of operations that must be supported by the payment processor when processing payment requests sent by the merchant. Payment operations depend, most of the times, in the nature of the payment method utilized. For example, for credit-oriented payment methods, like credit cards or letters of credit, there are three possible payment operations: authorize, deposit and credit. For other methods of payment, like electronic checks, follow a fire-and-forget philosophy where authorize and deposits are performed in the same step, in a single operation, and credit operations do not exist. Moreover, other forms of electronic operations like reversals (authorize reversal, deposit reversal and credit reversal) can also be supported, depending on the flexibility offered by issuing banks and payment providers.

Subscription: This policy determines how long a merchant has contracted the services of a payment provider. Typically the longer the contracted period the more advantageous it will be for the merchant - greater service levels and cheaper fees. Some examples of subscription policies are: Monthly, Quarterly, Yearly.

Associated Fees: This policy establishes the fee matrix contracted between the payment processor and the merchant. This policy can actually depend on other policies and negotiation between merchant and payment provider. The fee matrix may also be variable, for example, a higher fee for the first thousands of transactions in the subscription period, and a gradual decrease of fees as the number of transactions grow. Also not only per transaction fees exist. Other fees, like subscription fees, service fees, customer support fees, may exist and be adjusted according to the policies associated to the fee matrix.

Geography: The geography of the merchant may influence the terms of the contract between merchant and payment provider. Factors like local laws and risky geographies (for example, support for international transactions) are some of the factors that may influence this policy. Another factor often considered is round-the-clock processing support - for example, some geography can provide processing and customer support in off-peak times, while mainstream operations are not supported in the local geography. Also, sometimes payment providers have no support for payment processing originated from certain geographies.

Other QoS-related Policies: Several other policies can also contribute with considerable influence over the decision to use or not a certain payment service provider. Some of these policies are mentioned here as an example: contracted maximum amount of down-time, support for alternate and/or dedicated servers, daily versus seasonal off-peak support, etc.

As we can see, there are several different variables involved in the terms of service between merchants and payment providers. This list above is not exhaustive. Certainly other policies and variants will exist and they all can influence both the level of service and the costs associated with payment requests that must be processed.

2.2 An Autonomic Approach

The autonomic computing approach provides well established architectural concepts that can be used to model payment brokers as intelligent agents utilized as mediators between client software (the merchant applications) and server software (the payment processor system). In this approach we reference [IBMAC] and assume the following model mappings:

Merchant Application: The merchant application implements the interface with the end user in behalf of the merchant. In an online ordering system, for example, the merchant application will capture order and payment information using a graphical user interface available to the end consumers, or buyers, or an operator in the merchant's customer service desk. We equate this behavior with that associated with a Manual Manager in the Autonomic Computing architecture. The role performed in this application corresponds to the *Monitoring* layer of the autonomic computing control loop.

Payment Provider: The payment provider system implements the access to the resources, or the institutions and accounts holding the actual resources being managed, i.e. the accounts and funds in these accounts held at banks and other financial institutions. We equate this behavior with a Touchpoint in the Autonomic Computing architecture. The manageable resources are the accounts and funds being transferred between accounts. The manageable resource capabilities are the payment operations supported by the financial institutions.

Payment Broker: The payment broker agent implements intelligent behavior that otherwise would not exist in the interaction between merchant and payment provider. In this role, the payment broker performs optimization analysis and planning and tries to offer the best cost-benefit relationship between merchants and supported policies by payment providers. In addition, a payment broker can also establish the direct relationship with payment providers necessary to execute payment operations. We equate this behavior with that of an autonomic manager in the Autonomic Computing architecture. It implements the *Analysis*, *Planning* and *Execution* layers of the Autonomic Computing control loop.

Policy Repository: In this architecture, the payment providers must publish their business policies to common repositories that can be read by the payment broker. This collection of policies consists of knowledge that is transferred from the payment providers and can be utilized by the payment broker in the analysis and planning phases. We equate this policy repository with a Knowledge Source in the Autonomic Computing architecture. Different forms of knowledge may be stored in knowledge sources and transferred to autonomic managers to influence their behavior.

Several interactions are defined between these components. In order to be truly autonomic, the payment broker must capture payment provider policies from existing policy repositories. Payment providers first advertise their policies through a common mechanism, like a web service, and once this information is retrieved, it is published in these distributed policy repositories. Then, every time a payment broker wants to establish a relationship with a given payment provider, it looks up for the

correspondent policy repository that carries information about that particular payment provider and loads its policies into memory for analysis purposes.

Payment providers act as touchpoints and publish their capabilities along with their policies, but this information is not always stored in the policy repository. A capability query can be issued between payment brokers and payment providers. This can be implemented as part of the manageability interface of touchpoints, that follows the WS-DM MUWS standard – please refer to [MUWS] for more information.

Once payment brokers and payment providers are up and running and have established relationships (i.e. the payment processor policies are loaded into the payment brokers), the merchant applications can start using the analysis benefits provided by the payment broker architecture. Merchant applications perform the *Monitoring* layer of an autonomic manager, by capturing orders and payment information associated to these orders. Then, instead of sending payment requests directly to payment providers, these applications package their requests with merchant and order profile information and send them to payment brokers that will then be responsible by performing *Analysis*, *Planning* and sometimes also *Execution* of consolidated payment requests that now go to their end target, the payment provider. Upon successful completion, the payment provider sends payment results to their original merchant applications.

Analysis and Planning are two important steps performed by the payment broker that would otherwise not be performed in the traditional payment architecture. The following figure shows the elements of the architecture, their mapping into Autonomic Computing concepts, and the interactions between them.

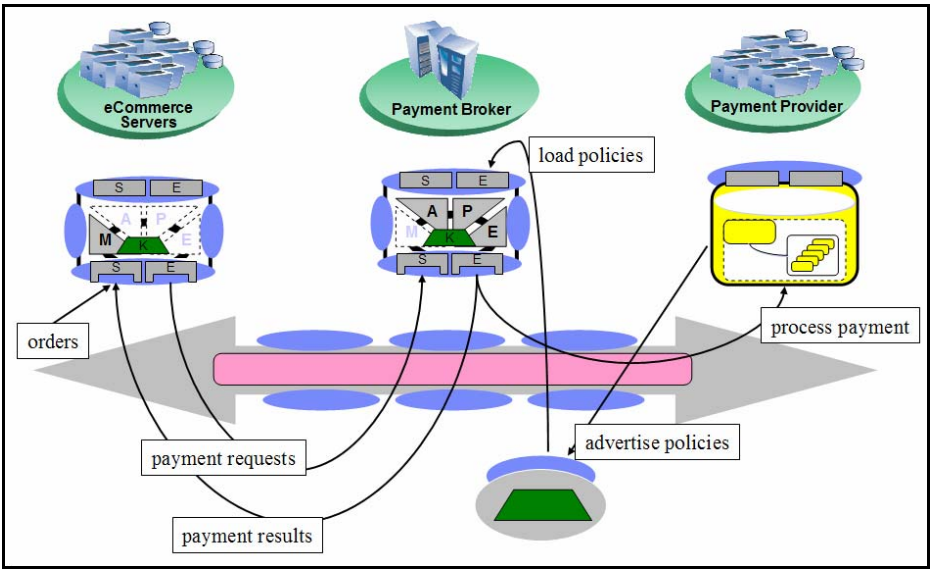


Fig. 3. Autonomic Computing concepts for the Payment Broker architecture

2.3 Payment Broker Internals

The payment broker performs three important tasks: *Analysis*, *Planning* and *Execution*, each with different goals. The combined goal is to provide the best cost-benefit ratio to payment provider customers, i.e. the merchant applications. With that goal in mind, savings in costs and quality can be passed onto the market. We now explain in more detail what activities are accomplished in each of these tasks.

Analysis: The analysis layer of an autonomic manager is where external stimuli captured in the Monitoring layer is drilled down and compared to internal knowledge in order to determine if a specific reaction is necessary or not. Most of the times, this reaction is the embodiment of aspects of the autonomic computing vision: self-healing, self-protecting, self-optimizing or self-configuring. In the payment broker the external stimuli corresponds to the payment requests sent by merchant applications along with merchant and order profiles. This information, along with specific knowledge (the payment policies published by payment providers) is utilized in heuristic analysis so the best ways to perform the requested payments are calculated. The merchant profile also contains merchant specific policies that contribute to the analysis, since it provides specific requirements on quality of service and associated costs that must be met. Moreover, multiple payment requests coming from the same merchant can be consolidated at this layer. Often, analysis performed in a large data space is more effective than if payment requests were analyzed one by one.

Planning: The planning layer of an autonomic manager is responsible by a broader scope analysis than that performed in the Analysis layer. Often a specific reaction is determined in the Analysis layer and carried over for validation and cause-effect analysis in the Planning layer. In this layer, the relationship among various manageable resources is taken in consideration, and a plan of execution is constructed. In the payment broker architecture, we do consolidation of payment requests originated from multiple merchants into bulks of requests that are then sent in batches to payment providers. This provides extra optimization and savings, since large amounts of payment requests often are advantageous to the customer - as previously mentioned payment providers often provide cost and quality incentives for customers with large transaction volume.

Execution: The execution layer of an autonomic manager is responsible by coordination of change plans created in the *Planning* layer and translating them into operations that are associated to touchpoints and managed resources. Alternatively, a standard payment interface could be developed and the translation to specific payment provider interfaces could be handled by touchpoints. It is our understanding that such standardization has been tried before and was not successful, due to the vast number of payment providers from different geographies and managing different types of payment. A more feasible alternative is to provide translation logic in the execution layer of payment brokers by supporting a plug-in architecture.

The following figure shows the layers and internal operation of payment brokers.

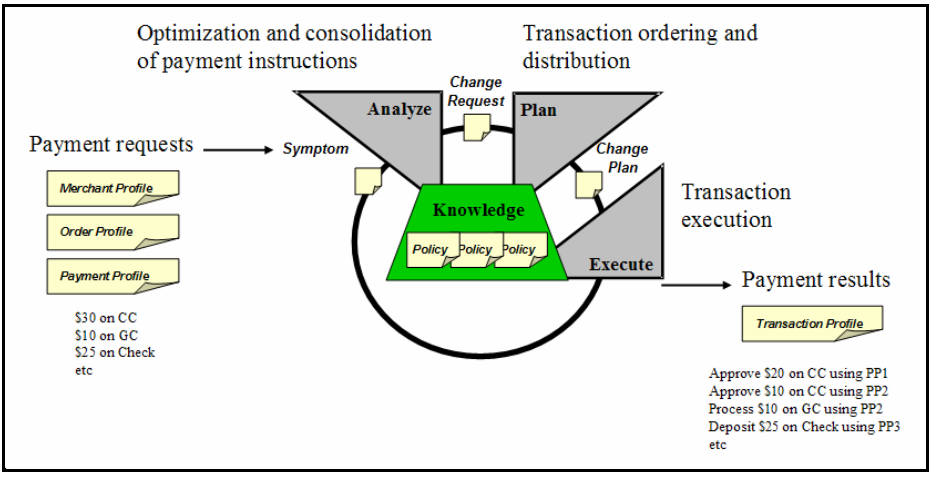


Fig. 4. Payment Broker Internals

As we can see in the picture the information that flows through each of these layers is based on standard knowledge formats, so different payment brokers could exchange this information between themselves in a standard way.

3 Case Study

In this case study we will show a very simple scenario where a payment broker could benefit merchants and help them achieve better costs and quality of service. Let's consider the following situation:

- Merchant M1 contracts Provider P1 to process its payment requests
- Payment Provider PP1 processes credit cards CC1, CC2 and CC3, and each payment transaction (for simplicity) costs 10 cents. Subscription costs $5K per quarter.
- Merchant M1 has an average volume of 300,000 transactions per quarter for each of CC1, CC2 and CC3.
- The associated cost will be $5K + $30K + $30K + $30K = $95K.

Suppose now that instead of having a direct contract with PP1 merchant M1 utilizes the services of Payment Broker PB1. A typical payment broker would consolidate several different merchants and payment providers together. For illustration purposes let's consider the following situation:

- Payment Broker PB1 contracts with 5 different Payment Providers: PP1 to PP5.
- Each Payment Provider charges $4K of subscription fees per quarter (a smaller subscription fee because of the higher transaction volume executed by PB1).

- Payment Provider PP1 charges 8 cents per transaction (again due to higher volume). Also, PP2 is a discount provider for CC1 and charges only 5 cents per transaction for that particular credit card (but charges higher amounts for CC2 and CC3).
- Payment Broker PB1 provides brokerage for 10 different merchants, each with the same average levels of M1.
- With some calculation we can see that in an optimal situation, Payment Broker could provide the same service to merchant M1 for a lesser subscription fee of only $2K ($4K times 5 providers divided by 10 merchants). In addition, the same level of service per transaction would cost only 5 cents for CC1 and 8 cents for each of CC2 and CC3. Processing 300,000 transactions for each of CC1, CC2 and CC3 would cost PB1 $15K + $24K + 24K.
- The total costs of the services of PB1 to M1 would then be $2K + $15K + $24K + $24K = $65K. Total savings of $30K would benefit M1 and allow it to lower its costs and prices.

In the real world payment transaction costs and associated quality of service levels may be much more complex than our sample scenario. As such, we are confident that significant savings in costs and quality of service can be achieved when payment brokers are fully implemented and provide a common access point to the millions of existing merchants that depend on complex payment provider services.

4 Conclusion and Future Directions

In this paper we presented a novel architecture that promotes intelligent analysis and consolidation of payment transactions. This allows for better cost-benefit ratios, lower fees and higher quality of service, to be offered to merchants.

We also proposed that with very simple heuristic analysis and planning, great savings and benefits on quality of service can be achieved.

Future directions of this work will now concentrate in the description and optimization of the analysis steps, so higher levels of benefit can be achieved. Given the complexity of payment provider policies and merchant requirements in the real world, heuristic analysis is not always sufficient for achieving optimal savings. Sometimes analysis of all existing variables is so complex that alternative techniques, like Learning and Adaptation can provide more efficient ways to achieve the same benefits and even to provide a higher level of accuracy. In a future work, we plan to implement different analysis mechanisms based on the self-monitoring of Payment Broker originated payment transactions and the injection of this data into a feedback loop in the *Analysis* and *Planning* layers. This historic transaction performance data will be used to fine tune the existing heuristics, so a higher level of benefits can be achieved. In the long run, additional heuristics could be discovered and be added into the main set of rules that govern the analysis of policies. With this approach we hope to bring maximum possible savings to merchants and their end customers.

References

[IBMAC] IBM Corporation, *"An Architectural Blueprint for Autonomic Computing"*, http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf, October 2004

[MUWS] OASIS Standards Consortium, *"Management Using Web Services"*, Parts 1 and 2, http://docs.oasis-open.org/wsdm/2004/12/wsdm-1.0.zip, December 2004

[UML] OMG – Object Management Group, *"Unified Modeling Language"*, Version 2.0, http://www.omg.org/technology/documents/formal/uml.htm

[MICRO] World Wide Web Consortium, *"Micropayments Overview"*, http://www.w3.org/ECommerce/Micropayments/

[MC] Payment Service Provider: *"MasterCard"*, http://www.mastercard.com/index.html

[VISA] Payment Service Provider: *"VISA3D"*, http://international.visa.com/fb/paytech/secure/ main.jsp

[SET] Payment Service Provider: *"SET – Secure Electronic Transactions"*, http://www.oiste.org/

[ACH] Payment Service Provider: *"ACH – Automated Clearing House"*, http://www.ffiec.gov/ffiecinfobase/booklets/Retail/retail_02d.html

[PAYPAL] Payment Service Provider: *"PayPal"*, http://www.paypal.com/

[CYBER] Payment Service Provider: *"CyberSource"*, http://www.cybersource.com/

[PARA] Payment Service Provider: *"ParaData"*, http://www.paradata.com/

[VS] Payment Service Provider: *"VeriSign"*, http://www.verisign.com/

[FIRST] Payment Service Provider: *"FirstData"*, http://www.firstdata.com/

[CITI] Payment Service Provider: *"CitiBank"*, http://www.citibank.com/

Genetically Modified Software: Realizing Viable Autonomic Agency

A.G. Laws, A. Taleb-Bendiab, and S.J. Wade

School of Computing & Mathematical Sciences
Liverpool John Moores University
United Kingdom, L3 3AF
{A.Laws,A.Talebbendiab,S.J.Wade}@ljmu.ac.uk

Abstract. Inspired by the autonomic aspects of the human central nervous system, the vision of "*autonomic computing*" arrived with a fully-formed wish list of characteristics that such systems should exhibit, essentially those self-referential aspects required for effective self-management. Here, the authors contend that the biologically-inspired managerial cybernetics of Beer's Viable System Model (VSM) provides significant conceptual guidance for the development of a general architecture for the operation and management of such complex, evolving, adaptive systems. Consequently, the VSM has been used as the basis of a theoretically-supported reference model that provides the "blueprint" for an extensible intelligent agent architecture. Of course, normal use of the VSM relies heavily on human agency to realize the adaptive capabilities required by the model. Therefore, artificially replicating such activities represents a significant challenge, however the authors show that some progress can be made using algorithmic hot swapping and in particular Holland's Genetic Algorithms (GA's) to generate, in specific circumstances, a repertoire of tailored responses to environmental change. The authors then speculate on the use of the associated Learning Classifier Systems (LCS) approach to allow the system to develop an adaptive environmental model of appropriate, optimized responses.

1 Introduction

Managerial cybernetics, rooted in the study of the adaptation of human organizations in a changing environment, underpinned by classical cybernetics and structured around the human central nervous system have long been used to study the viability of human systems [1]. More recently and with the advent of the possibility of "*autonomic computing systems"* realizable by *adaptive software agents*, the need for a unifying theory to inform and guide the development of such systems has become apparent. It would appear that cybernetic studies have much to contribute in terms of a well-established foundation for the development of such systems. In this paper, the authors attempt to show both the drawbacks and value of assuming such a position by demonstrating the contribution that a cybernetic viewpoint can bring to the development of such systems.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 184 – 196, 2006.

The remainder of this position paper is organized as follows. The next section provides a brief introduction to the managerial cybernetics of Beer's Viable System Model (VSM) [1]. This is followed by the development of a theoretically defensible adaptive software architecture realizable by means of multi-agent software paradigm [2]. Some drawbacks inherent in transferring the VSM from its normal human-oriented use to the artificial domain of software systems are noted. The next section addresses these concerns by reviewing the classical cybernetics that underpin the VSM to demonstrate the appropriateness of the use of genetically inspired approaches to provide our target system with elements of "creativity" and "learning". Such approaches are considered in the penultimate section of the paper, which then concludes by both acknowledging the inherent limitations of such approaches and speculating on the further research opportunities these nevertheless afford.

2 The Viable System Model – An Architectural Model for Self-managing Systems

The Viable System Model [1], founded on classical cybernetic studies and modelled on the human central nervous system, provides a theoretically supported cybernetic model of organization. Viable systems may be defined as being robust against internal malfunction and external disturbances and have the ability to continually respond and adapt to unexpected stimuli. The model specifically attempts to imbue the system with the ability to adapt to circumstances not foreseen by the original designer and identifies the *necessary* and *sufficient* communication and control systems that must exist for any organization to remain *viable* in a changing environment. The major systems (i.e. S1s, S2s, S3, S3*s, S4 and S5) are structured hierarchically and connected by a central 'spine' of communication channels passing from the higher-level systems through each of the S1 management elements (Fig. 1.). These provide high priority communication facilities to determine resource requirements, accounting for allocated resources, alerts indicating that a particular plan is failing and re-planning is necessary and the provision of the "legal and corporate requirements" or policies of the system.

The systems shown in Fig. 1 concern the management structure at one level of the system, and consequently specifies the communication and control structures that must exist to manage a set of S1 units. However, the power of the model derives from its recursive nature. Each S1, consisting of an operational element and it's management unit, is expected to develop a similar VSM structure, consequently, the structure of systems is open ended in both directions and may be pursued either upwards to ever wider encompassing systems or downwards to ever smaller units.

However, at each level the same structure of systems would occur although their detail would necessarily differ depending on context. This recursivity allows each level in the organization relative autonomy bounded by the overall purpose of the system as a whole. Such cooperation and coordination within and between S1 units on the same hierarchical level and between sets of S1's on different levels is provided by communication channels operating over an organizational range and tailored locally to each viable entity.

However, if the recursion is pursued below the level of human operations then it is possible to consider the software system as the next recursive level in the system. This strongly implies that the software system should assume the same overall management structure as the human process above it. Consequently, we appear to have a theoretically supported management architecture both for the human process and for "self-managing autonomic software"

The value of assuming such a viewpoint is in the immediate provision not only of the outline architecture that the autonomic software system itself must assume, namely that of the Viable System Model, but also the identification of the requisite communication links to bind the system to the organization.

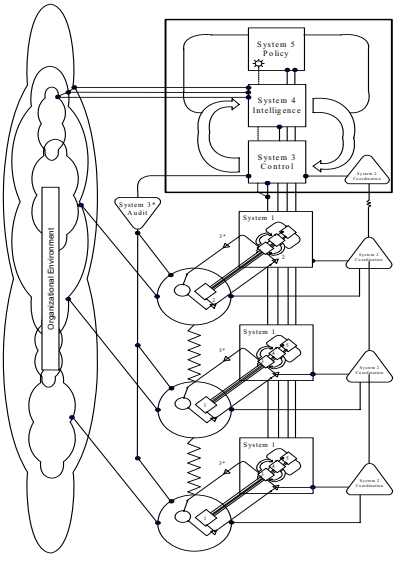


Fig. 1. The Viable System Model

3 An Intelligent Agent Architecture

We now extend and apply this cybernetic approach and consider an S1 of the VSM in terms of an autonomic software system [2]. To demonstrate, a conceptual, architectural outline of such a system is determined, using both the principles of the VSM and the terminology and design of a classical Artificial Intelligence design, namely Bratman *et al.*'s Intelligent Resource-Bounded Machine Architecture (IRMA) [3] as a constructional guide. As shown in Figure 2, the developed VIAA/J-Reference architecture embeds a Beliefs, Desires, Intentions (BDI) unit at the S5 level representing:

- Desires - or what the agent wants to do and is taken as a given for the moment.

- Beliefs - or what the system currently knows and is represented by two structures. A model of the external world and a model of the current internal status of the architecture.
- Intentions - or what will actually be done, is determined by a process of deliberation, which interprets desires in the light of current beliefs about both the environment and the 'stance' of the system.

S3, using a reasoning process supported by a plan library and the capacity to audit the current status of operational S1 units, structures the intentions into plans, these are then passed to a scheduling process. The scheduling process, in cooperation with a resource bargaining process, responsible for negotiating resource deployment and usage monitoring, schedule the enactment of the plan. The schedule passes to the coordinating S2 channel for dissemination to participating S1 elements.

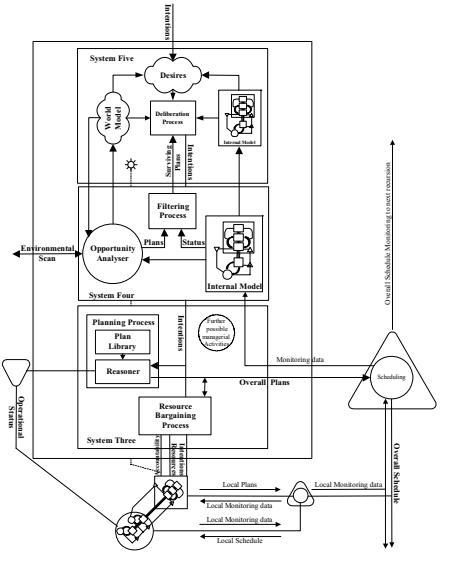


Fig. 2. The J-Reference Model

Environmental change is addressed by S4, which equipped with an Opportunity Analyzer and guided by the S5 desires model, scans the environment for detrimental events or beneficial opportunities. There are two outcomes of this process; the first is the formulation of a view of the outside world which is provided to S5 in the form of the World model. The second outcome is the production of development plans for the future of the system, either exploiting advantageous opportunities or avoiding detrimental occurrences. Plans are then passed to the deliberation process to begin the intention forming cycle again.

As noted above, the power of this approach lies in the recursivity of the underlying model. Figure 2, indicates that the entire architecture described above is repeated in the client S1 unit in the next layer. Consequently, the intentions channel at one

recursion informs the desires model in the next, thus allowing an autonomous response to local conditions at each level while remaining within the purpose of the overall organization.

While basing such a model on the VSM is an attractive proposition, it is not without cost. The VSM, designed for use in human systems, relies in no small part on the human agents that populate the model to provide the creativity and learning necessary to realize many of the systems advocated. For example, the environmental scanning and construction of a model of the environment required by System 4 is relatively straightforward for humans but notoriously difficult to achieve in software systems. While such modelling may be achieved by software systems in relatively small, finite environments, the more open the environment is the more difficult the task becomes. However, much conceptual guidance in addressing these problems may be derived from the classical cybernetics that underpin the VSM.

4 Classical Cybernetics Revisited

While the managerial architecture presented above provides an overall structure for the system, classical cybernetics provides the theoretical foundation for the detail needed to fill out that framework. Here, we consider the more general lessons learned from the cybernetic study of adaptive organisms surviving in a changing environment. Central to this approach is the notion of system state, where both the organism under consideration and the environment in which it exists are described by sets of variables that indicate the current status of each party. Consequently, the environment of the organism can be defined as:

> *"...those variables whose changes affect the organism and those variables that are changed by the organism's behaviour."* [4].

Similarly, the environmental changes that the system is capable of responding to effectively is measured by the number of states that the system can adopt, this measure is termed *variety,* where variety is defined as:

> *"the number of possible states of a system."* [5].

This leads to the view that one form of adaptation is aimed at the maintenance of the internal stability of the organism as represented by the state of its critical variables or as Ashby puts it:

> *"...a form of behaviour is adaptive if it maintains the essential variables within physiological limits."* [4].

To accomplish this, the system must have the ability to influence or cause change in other elements that make up the environment. In effect, the system must attempt to exert a form of “control” over some part of the environment. However, control can only be attained if the variety of the controller is at least as great as the situation to be controlled. This is Ashby's *Law of Requisite Variety*, which simply stated indicates that for every environmental action there is an equal and opposite response [6]. Moreover, given a range of possible responses to an environmental disturbance, the system must “know” which action to select from a variety of available actions; this is

the "*Law of Requisite Knowledge"* [7]. Furthermore, in order to ensure effective control or regulation of a controlled situation requires that the controller model the situation to be controlled. This requirement, known as the Conant-Ashby theorem, states:

"Every good regulator of a system must be a model of that system." [8].

So ideally, an optimal regulator will be an isomorphic model of the situation to be controlled. However, when such isomorphism is not possible as in highly complex systems, then the regulator must be, i.e. contain, a homomorphic model of the situation.

5 Algorithmic "Hot Swapping", Algorithm Generation and Learning Classifier Systems

Equipping our system to conform to the Law of Requisite Variety requires either the provision of a response for every possible environmental change, the provision of the ability to generate a response or perhaps a mixture of the two. The first option is limited in application as it requires the software designer to predict and cater for every eventuality the system may encounter and adherence to a consistent terminology/ontology, etc. The second option, that of algorithmic generation, whilst even more challenging, offers the potential for significant advances in the nature of systems development.

5.1 An Experiment in Algorithmic "Hot Swapping"

In our first attempt to address Ashby's Law of Requisite Variety, we first define an environmental scenario to which our system must respond. Here, the environment is considered to provide a supply of $n = 5$ element arrays of integers for sorting. This approach allows for a closed, highly controllable environment that can range from smooth, i.e. relatively small changes between arrays to a highly discontinuous environment where subsequent arrays may vary between almost sorted to complete reversal. The system is provided with a finite set of responses to environmental change and is charged with the task of matching the most appropriate, i.e. efficient response to the current environmental position at runtime. Here, the response set is represented by a library of sorting algorithms, each of which is capable of sorting any array received from the environment, although not necessarily optimally. To facilitate the system in its task, a means is provided that allows the system to determine the efficiency of the responses at its disposal.

In these experiments, the system was provided with the means to time an algorithm in execution and hence determine the most efficient algorithm in a given situation. This was achieved using the "Zen Timer" described by Abrash [9] and provided as a freely available "C" callable library capable of timing code fragments with micro-second accuracy. Running the system on a 40 MHz, 486 machine allowed for very small, highly controllable, 5-element arrays to be used while retaining the timing discrimination necessary to determine efficiency.

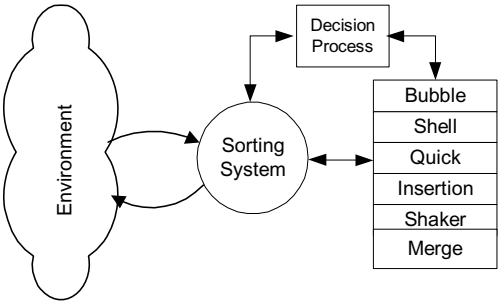


Fig. 3. The environment presents the system with 5-element arrays for sorting. Environmental change is represented by degrees of unsortedness in the arrays. The system, provided with a set of responses, must determine the optimal response.

Consequently, the system "experimented" to determine the most efficient response to the current environmental stance. Assuming that the environment was changing relatively slowly, the selected algorithm was used for subsequent arrays delivered by the environment. In order to ensure the currently selected response remained the most appropriate, a statistical process control approach was adopted [4].

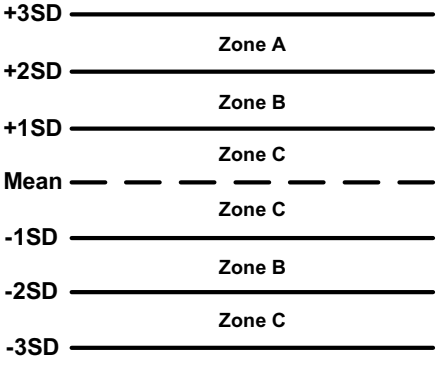


Fig. 4. The Western Electric Statistical Boundaries

Using the Western Electric Rules approach, statistical boundaries were set at plus and minus 1, 2 and 3 standard deviations from the current mean measure as shown in Figure 4 above [4].

The following rules were then applied to trigger a further experimental "review" of the timed efficiency of the current response.

- **Rule 1:** A single point falls above or below three standard deviations (beyond Zone A).
- **Rule 2:** Two out of three successive points lie in Zone A on one side of the mean.

- **Rule 3:** Four out of five successive points fall in or beyond Zone B on one side of the mean.
- **Rule 4:** Fifteen points in a row fall in Zone C on either side of the mean.
- **Rule 5:** Eight points in a row fall on one side of the mean.

The resulting control chart is shown below in Figure 5.

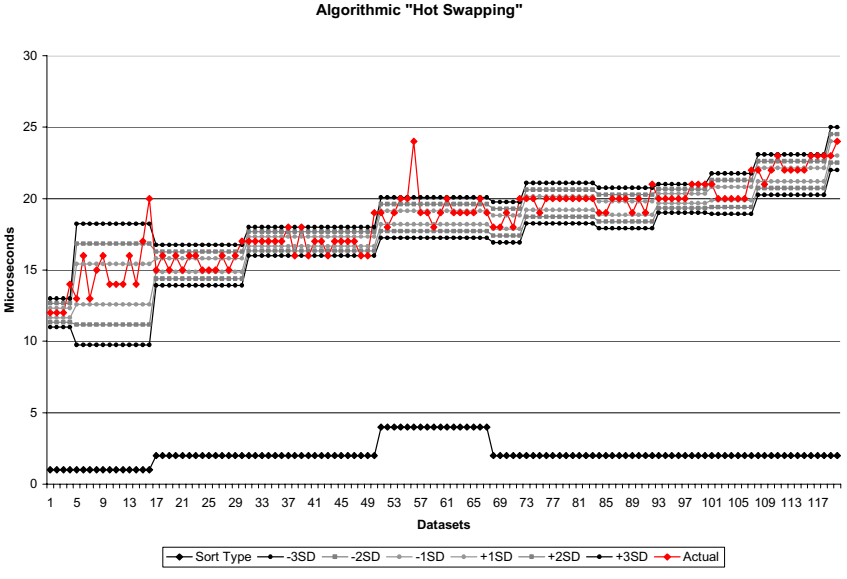


Fig. 5. Algorithmic "Hot Swapping" using statistical process control

The lowest line on the chart indicates the sort algorithm currently selected, where 1 = Bubble sort, 2 = Shell sort, 3 = Quick sort, 4 = Insertion sort, 5 = Shaker sort and 6 = Merge sort, although Quick, Shaker and Merge are never selected in this experiment.

While algorithmic "hot swapping" provides the system with a limited degree of adaptive capability, the finite nature of the set of responses constrains the degree of optimisation possible. In many respects this approach is akin to a programmer predicting the circumstances a system may have to contend with and providing a response to each circumstance. A more interesting question is whether a system can generate its own set of responses.

5.2 Algorithmic Generation

In constructing an approach to this question, a range of genetically inspired approaches have been considered. Beginning simply, a repetition with some variation of Hillis's well-known experiments in designing an optimal $n = 16$ sorting network using a genetic algorithm [11] and (incidentally) a Connection Machine [12], as described by Mitchell [13] was undertaken. Here as above, the environment is considered to provide an infinite supply of n element arrays of integers for sorting.

The task is to evolve an appropriate/optimal sorting mechanism using a genetic algorithm. In our experiments $n = 8$ was used although a Connection Machine was not. The manner of encoding the problem and hence the design of an appropriate genetic algorithm closely followed that developed by Hillis [12] with obvious reductions of scale to account for our more limited experiments.

In this approach, a sorting algorithm is simply considered as a series of pairs of array indices to be compared with the values held in those locations being swapped if appropriate. Consequently, the list (0, 6), (3, 1), (2, 4)…represents "*Compare the value in array location 0 with the value in array location 6 and swap if necessary and so on…"*. Encoding comparisons to form a sorting algorithm employed ten 12-bit chromosomes. Each chromosome being composed of four 3-bit "*codons*", with each 3-bit codon being capable of holding a binary value between 0 and 7 and consequently capable of uniquely identifying one location on an $n = 8$ array.

Each 12-bit chromosome therefore holds four codons and hence two array location pairs for comparison, e.g. (3, 4) (1, 5). Using ten such chromosomes allows twenty comparisons to be stored in total and represents one sorting algorithm. Such an encoding represents, in the terminology of genetic algorithms, the *genotype* or genetic constitution of an individual in a population, decoding that genotype and hence constructing the individual forms the *phenotype* or actual representation of the individual. All actions are performed on the genotype and reflected in the characteristics of the resultant phenotype.

However and again following Hillis, when the genotype is decoded to form a phenotype, rather than simply reading through the ten chromosomes and listing the comparisons stored there, a *diploid* approach is used whereby chromosomes are considered in pairs [5]. So, for each of the five pairs of chromosomes in a genotype, the comparison pair stored in codons 1 and 2 of chromosome A are compared with those stored in codons 1 and 2 of chromosome B. If both encode the same comparison (i.e. are *homozygous*), then only one comparison is realized in the phenotype. If, on the other hand, each codon pair in each chromosome encodes different comparisons then both appear in the phenotype. The comparison then continues with codons 3 and 4 of both chromosomes A and B. Eliminating duplication in this manner means that the size of the resulting phenotype can vary from a minimum of ten comparisons representing maximum duplication, to a maximum of twenty comparisons representing no duplications. Allowing variation in the size of algorithms that emerge in terms of numbers of comparisons and possible swaps undertaken has obvious implications on the efficiency of the resulting algorithms.

In our experiments, a population of 200 members, each with the above encoding, was formed on a 10 x 20 two-dimensional array. The codons of each chromosome being populated by a coin-tossing algorithm producing random bits. Each member of the population was then tested against 100 test cases drawn from the 40,320 permutations possible for $n = 8$. Although no member of the initial population was "correct" in terms of being able to correctly sort all 100 test cases, happenstance demands that even a random set of comparisons followed by swapping if appropriate will occasionally result in an improvement in the degree of "sortedness" of an initially unsorted array. Consequently, the "*fitness*" of each member of the population was determined by its ability to reduce the degree of "unsortedness" in the test set. The inversions method described by Knuth [10] was used to determine the degree of

"unsortedness" in a test array before and after its exposure to a sorting algorithm and hence identify any decrease or otherwise.

Having determined the fitness of each individual in the population, the average fitness was computed and those individuals with lower than average fitness culled. The resulting gaps in the population were filled with copies of high fitness individuals.

Next, each consecutive pair of individuals in the population contributes one offspring to the next generation. As before, a diploid approach was adopted for reproduction. Here, each of the five pairs of chromosomes that make up each parent is considered. A random "*crossover*" point is determined for each pair of chromosomes. A new "*gamete"* is produced by taking the chromosomal elements before the crossover point of chromosome A and joining them to the chromosomal elements after the crossover point of chromosome B to form a new 12 bit chromosome. Thus, each parent contributes five such gametes to the production of a new 10 chromosome member of the next generation. These reproductive activities continue until a new population of 200 newly-formed members is created. Mutation is then applied to the new population by randomly flipping bits in the randomly selected chromosomes of randomly selected individuals using the standard mutation probability of 0.001. Finally, once the new population is complete, fitness testing can resume as before.

5.2.1 Results

Our experiments, initially run over 100 generations, showed that a sorting algorithm capable of correctly sorting all of the test cases emerged surprisingly quickly. As can be seen in the truncated chart shown in Figure 6 below, the first fully-fit individual with a fitness rating of 1 emerged in generation 5 in this particular run. This result was replicated in further runs of the experiment with only minor variations in the generation that the individual emerged.

Admittedly, these results were obtained using a relatively gentle environment of test cases, running from the "*already sorted*" test case represented by the test array "1, 2, 3, 4, 5, 6, 7, 8" with an "unsortedness" rating of 0[1] to the "*partially sorted*" case of "1, 2, 3, 7, 8, 6, 5, 4" and an unsortedness rating of 9. Using a more severe test set running from an unsortedness rating of 19 and represented by the test array "8, 7, 6, 2, 1, 3, 4, 5" to an unsortedness rating of 28 representing a "*complete reversal*" of the array "8, 7, 6, 5, 4, 3, 2, 1" caused the system some problems, with no fully-fit individual emerging although 97% of test cases in this set were correctly sorted.

These shortcomings notwithstanding, a number of interesting points emerge from these experiments. The fact that a correct algorithm emerged so quickly, even in such small scale, benign experiments was surprising. Arguably, once a fully-fit individual is identified, an appropriate response to, at least this region of the environment, has been rapidly discovered and the process could terminate. However, in the course of the remaining generations, the system continued to "discover" new approaches to dealing with these test cases, with a variety of algorithms emerging displaying greater or lesser homozygous duplication. One fully-fit individual emerged that used the minimum number of comparisons, i.e. ten; to sort the test cases, while others were discovered using a range of comparisons between 10 and 18.

[1] Using the inversions method described by Knuth.

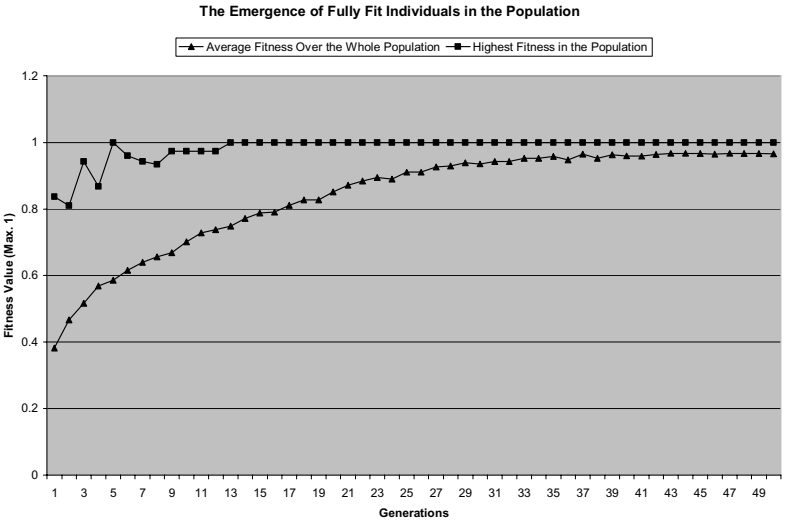


Fig. 6. Fully-fit individuals attributed with a value of 1 and capable of correctly sorting all 100 test cases emerge very early in the generation process

This implies that the system could quickly derive an appropriate response to an environmental region while continuing to search for more optimized responses. Indeed, in these experiments, no explicit selective pressure was used to develop minimal, correct sorting algorithms, although clearly it could be. This, of course, raises the possibility of automatically deriving a portfolio of algorithms each optimized for a specific area of the environment and offering enhanced sorting performance over a universal sort algorithm.

Of course, except in specific circumstances, such an approach is clearly unsuitable for general application. Even the encoding used in the experiments reported here demonstrates the fragility of the approach. Here, a 3-bit codon randomly filled with bits always results in an integer value between 0 and 7 and can subsequently be used to access an 8 element array. It is not so easy to make such arrangements for a 10-element array. Nevertheless, such experimentation helps to determine the limits of applicability of such approaches and lays the foundation for further work in associated fields such as genetic programming [14] or Artificial Immune Systems (AIS).

Having generated a repertoire of responses, it is still necessary for the system to develop the appropriate knowledge to use each response appropriately, i.e. conform to the Law of Requisite Knowledge. Holland's Learning Classifier Systems (LCS) [11] provide a means whereby such knowledge may be developed. Such systems are constructed using; detectors to convey the state of the environment to the system, a rule and message system where messages are used both for internal processing and to direct the system's effectors that act upon the environment, an apportionment of credit system that, using some measure of performance provided by the environment, apportions that credit to participating rules and a genetic algorithm to develop new

rules. Rules are formulated in `if <condition> then <action>` sets and those that match the condition reported by effectors compete to contribute in the final action reported by effectors. Rules accrue or lose reward depending on the success or otherwise of the outcome reported by the environment. New rules can then be genetically derived from already proven effective rule sets. What emerges is an adaptive model of the environment of the system encapsulated in effective rules, essentially the system *learns* to adapt to the environment in which it resides.

6 Conclusions

In this paper, the authors have presented a conceptual architecture derived from managerial cybernetics and suitable for the operation and management of autonomic software systems. While noting the drawbacks inherent in applying a human-oriented model to artificial systems, it has also been shown that the theoretical basis of the model is resilient enough to point to existing techniques that may be used to begin to address these shortcomings and progress this work.

What emerges is, in some respects, a unifying, hybrid architecture that populates the structured, model-based architecture implied by the J-Reference model, with the "swarm-based" nature of genetic approaches. Of course, much remains to be done, the genetic experiments reported here suggest that using selective pressure to produce minimal sorting algorithms may have interesting results and the use of genetic programming promises a much more generalized approach. Similarly, the LCS approach appears to support the development of incomplete though adaptive models of the environment that lay the foundations for more sophisticated uses such as anticipatory system pre-adaptation.

References

[1] Beer, S., *Brain of the Firm*, 2nd ed, John Wiley & Sons, Chichester, (1981).

[2] Laws, A.G., A. Taleb-Bendiab, S.J. Wade, and D. Reilly, *From Wetware to Software: A Cybernetic Perspective of Self-Adaptive Software*, in *Self-Adaptive Software: Applications, Second International Workshop on Self-Adaptive Software*, R. Laddaga, P. Robertson, and H. Shrobe, Editors, Springer-Verlag: Berlin.(2003) pp.

[3] Bratman, M.E., D.J. Israel, and M.E. Pollack, *Plans and Resource-Bounded Practical Reasoning,* Computational Intelligence, Vol. **4**, No. 4, (1988), pp.349-355.

[4] Ashby, W.R., *Design for a Brain*, Chapman & Hall, London, (1954).

[5] Espejo, R. and R. Harnden, *The Viable Systems Model - Interpretations and Applications of Stafford Beer's VSM*, John Wiley & Sons, Chicester, (1989).

[6] Ashby, W.R., *An Introduction To Cybernetics*, Chapman & Hall, London, (1956).

[7] Heylighen, F. and C. Joslyn, *The Law of Requisite Knowledge,* http://pespmc1.vub.ac.be/REQKNOW.html,(1993).

[8] Ashby, W.R., *Every good regulator of a system must be a model of that system,* International Journal of System Science, Vol. **1**, No. 2, (1970), pp.89-97.

[9] Abrash, M., *Zen of Code Optimization*, Coriolis Group Books, (1994).

[10] Knuth, D.E., *The Art of Computer Programming: Sorting and Searching*. Vol. 3, Addison-Wesley Publishing Company, Reading, Mass, (1973).

[11] Holland, J.H., *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, Mass, (1992).
[12] Hillis, W.D., *Co-evolving parasites improve simulated evolution as an optimization procedure,* Physica D, Vol., No. 42, (1990), pp.228-234.
[13] Mitchell, M., *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, Mass, (1996).
[14] Koza, J.R., *Genetic Programming II: Automated Discovery of Reusable Programs*, The MIT Press, Cambridge, Mass, (1994).

Harnessing Self-modifying Code for Resilient Software

Christian Tschudin and Lidia Yamamoto

Computer Networks Group, Computer Science Department
University of Basel, Bernoullistrasse 16, CH–4056 Basel, Switzerland
{christian.tschudin,lidia.yamamoto}@unibas.ch

Abstract. In this paper we argue that self-modifying code can become a better strategy for realizing long-lived autonomous software systems than static code, regardless how well it was validated and tested. We base our discussion on three facets – self-repairing software, adaptive software and networked systems – for which we point out ongoing and related work before presenting a roadmap towards a controlled framework for self-modifying code.

Keywords: Resilient software, self-healing protocols, computational agents, autonomic communication.

1 Introduction

In a famous article series in Scientific American from 1984, Dewdney introduced the "core wars" game [1]: Imprisoned in a small memory bank, two programs fight against each other by peeking and poking into the adversary's memory cells and by taking care of retaining the own software's integrity through repair actions or dislocation. On the hardware side, von Neumann considered unreliable logical gate implementations and showed in 1956 how to use redundancy to achieve fault tolerance despite defective execution elements [2].

These examples illustrate the simple idea that the road towards robust software starts with the humble assumption that no system or software is perfect and therefore any of its parts may fail. While mission-critical systems have considered robustness and fault-tolerance techniques at the core of their design, solutions are still lacking for light-weight and widely deployable *autonomous software* that is resilient and able to survive unsupervised in a large spectrum of situations.

One can imagine scenarios requiring autonomous distributed systems, where large networks of small devices must run unsupervised during a long period of time, such as sensor networks and embedded networks of pervasive computing devices, but also constellations of spacecrafts sent to distant outer space regions where ground control from Earth becomes impractical even for non real-time actions [3,4].

Two notions are combined here: adaptability and resilience. Ideally, a fully autonomous software system system should continuously update itself, recovering from failures and adapting to new situations. Since no realistic software

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 197–204, 2006.

system could possibly anticipate all future situations, such updates necessarily include changes to the system's own code base. Today these changes are largely performed off-line either by humans or in a semi-automated way with the help of software engineering tools able to generate parts of the code. Hardware oriented hardening techniques have received continued attention since von Neumann's work – most recently in quantum computing where inevitable execution noise has to be compensated through error correcting quantum execution circuitry. On the other hand, software oriented solutions have targeted fault tolerance by adding instead of changing components: A *fully* software based solution that considers memory, communication and execution errors and which provides self-healing capabilities is still lacking. We believe that self-modifying code, although banned from computer science because it is difficulty to debug and to assess its correctness, will play an important role in this quest.

Before discussing a roadmap towards a better mastering of self-modifying adaptive and resilient software, we show in three fields (soft errors, software adaption, mobile code) where software resilience has been envisaged and how self-modifying code comes into the picture.

2 Handling Soft Errors in Software

Soft errors are transient faults due to radiation (particles discharging a semiconductor capacitor) or other low level physical phenomena (noise in a quantum computing device). The effect is that either state is altered or instructions are incorrectly executed. An important counter measure against such incidents is to apply systems-level i.e., hardware changes. For example, memory cells store data in an error correcting code format that can revert a limited number of bit flips. Similarly, quantum computing circuits have been proposed that permit error correcting execution, hence masking away low level errors.

However, it is also possible to mitigate or even neutralize soft errors by software techniques alone. In [5], a compiler is described that schedules redundant instructions such that additional comparison instructions can detect whether an execution thread encountered an error or not. This code transformation approach relies on a reliable memory subsystem, but other proposals have been made that also include memory in the "sphere of replication". The SWIFT approach in [5] targets bit failures and applies to programs that can be recompiled. Low-level errors are again masked away and the program itself is not made aware of them. A small time window remains just after validation where validated state could be corrupted unnoticed, hence the sequential character of the execution environment becomes visible. Another weak point is that (erroneous) store operations are not observable.

Clearly, these software-implemented fault tolerance techniques can be applied to a software system at design time. We note however, that at run-time, the probability of fatal failure after an uncatched soft error (e.g., because too many bits were flipped etc) is high as the program itself relies on the installed safe guards. Such unnoticed errors will accumulate if no other self-checking mechanisms

are put into place. In addition to run-time only detection of execution errors, a software system should periodically verify its integrity and, if necessary, regenerate its own code base and memory (self-healing). This is further complicated if the software is capable of adapting to new situations (see next section), or is extended with imported functionality (see section 4 on networking), so that there is no constant reference point to which the current state of the system could be compared to. Overall, changed, imported or regenerated functionality are all special cases of self-modifying code, although at different levels of programmability[1].

3 Beyond Planned Adaptation–Run-Time Software Evolution

The ability to adapt to new situations is an essential property of any system that is expected to operate unsupervised. Adaptive software systems capture information from their environment and make changes in their internal structure in order to achieve a desired performance in that environment. These changes are performed according to an adaptive plan [6], which determines the transformation operations to be applied to the current structure to obtain a new one.

No matter how comprehensive an adaptive plan might be, in terms of covering the space of possible combinations of environment state variables, there will always be cases that have been left out of the plan, or for which the plan is not sufficiently reactive. Those cases generally call for humans to intervene and upgrade the plan.

A change in the environment that had not been predicted at design time causes one or more system components to become imperfect because they are no longer well adapted to the task at hand. The system must then diagnose and correct these imperfections, making the components suitable again. The same behavior leads to both adaptability and resilience. The central question is then how to automate this diagnose and repair process, not only for the system configuration but also for the running software. In other words, how to build software that genuinely evolves to correct itself and to match new expectations.

Structures that are subject to adaptation also include programs. A common technique applied to adapt programs is genetic programming (GP), in which programs are modeled as genetic material (genotype) and the adaptation plan consists in the way to apply genetic operations (e.g. crossover, mutation) to a population of programs to obtain new ones. GP can potentially cover a large environment space with a very simple plan, due to its problem-agnostic adaptive plan and high parallelism of solution search. However, it is slow to react due to the random nature of the code transformation operations, which cannot

[1] Note that the SWIFT approach collides with self-modifying code (at the level of CPU instructions): the compiler will not be able to apply appropriate resilient code transformation at compile time as the code to protect will only become available at run time.

ensure viable programs. Non-viable programs can only be detected after they obtain a low fitness value during execution. This limitation has prevented GP from being applied to running programs. However, in a resilient system as the one we describe below, invalid programs could be detected and eliminated with minimum impact in the system. GP would then become feasible in these systems, and we have started to explore this path in [7]. The question remains open whether improved genetic operators could be found to accelerate the search and to increase the likelihood of viable results, while at the same time keeping the plan to a minimum.

Formal techniques have been applied to generate code from specifications [8,9], however the focus has been on software engineering where tools are not fully automated and not directly applicable to running code. The adaptive plans resulting from these techniques are likely to become complex.

The quest for minimum plans is directly related to the quest for fully resilient software, with no "fixed" parts that cannot be adapted. The plan is one such fixed part, since the plan itself is not part of the structures subject to adaptation. Although GP plans are already small, researchers have taken steps towards reducing them even further by incorporating GP parameters into the genotype of the individuals being adapted [10], therefore including part of the plan into the adaptive structures themselves.

4 Networking and Mobile Software

Now consider the case of self-modifying distributed software, such as a distributed system or a network protocol stack. One can easily anticipate the difficulties in keeping the different pieces of the software fully functional and compatible, such that they can run in a consistent way to provide the intended service. Once a new software has been generated, it must deploy itself, i.e. ship its own constituent parts to the regions of the network where they are necessary. Code mobility is therefore a natural and essential ingredient of distributed and adaptive software.

The desired joint resilience/adaptability property discussed so far for a single system needs to be extended to the distributed context as well, creating robust distributed execution circuits that can recover from several disturbances such as message loss, losses of parts of their code base in different parts of the network, transient inconsistencies and temporarily unsuitable code portions.

Attempts on code mobility include active networking and mobile software agents. The main aspect of these approaches is the late binding of functionality which permits the adaptability to network conditions and user requirements at run time. Because they focus on the run-time-deployment of new (but fixed) functionality, existing mobile code systems like [11,12] cannot be easily used to build the robust self-healing execution circuits that we are aiming at. The main reason for that is the lack of access to the representation of the code itself, which would permit self-modification. Mobile code systems like M0 [13] have been proposed where a program is able to access the representation level of mobile

code units, leading to the possibility to create (compute) new code pieces at run-time. Such an expressiveness permits a much deeper degree of self-modification and enables otherwise not implementable protocols like e.g., self-compressing or self-fragmenting mobile code units. Moreover, because of safety concerns (and the implicit virus-like nature of mobile code), programming languages and execution environments like [11,12] have been deliberately restricted which removes the ability to write self-modifying mobile programs.

More advanced yet more secure functionality based on self-modifying code could be permitted if there was a methodology to generate safe self-modifying programs. One currently hypothetical option would be to design an environment for mobile *specifications* (instead of mobile code) that can capture the essence of self-modification but leaves the actual implementation to a safety-property preserving run-time code generation mechanism. In the following section we come back to this aspect of run-time code generation, where code is not only generated once but every time before it is executed.

5 A Roadmap for Self-modifying Code

We see three related research challenges where more insights for using self-modifying code are needed:

- execution models,
- software regulation techniques, and
- self-modifying code generation methods.

A crucial element and research target for resilient software is an appropriate *execution model* and the question how such an environment supports self-modifying code. The execution model encompasses the properties of the memory, the execution part as well as the communication primitives. From results in fault tolerance it seems that the parallel execution of activities is an essential element of an execution model. However, first results (discussed in Section 2) show that soft error resilience can also be achieved inside a single-threaded sequential execution environment (although preventing self-modifying code). Regarding the effective implementation of resilient software, more research is needed to understand the requirements on the execution model.

In our research so far we have been working with tiny computational agents that we call Fraglets [14]. Fraglets are concurrent computation fragments similar to chemical molecules such that they can operate on each other. Code and data are processed by reactions between Fraglets, thus putting code self-modification at the heart of the execution model.

In Figure 1 we present a Fraglet inspired execution model that shows essential processes for a purely software-based resilience approach. The software system is placed in a hostile environment where memory faults (soft error, but also loss of memory content due to programming errors or contention for memory), execution errors (altered code, either transiently or permanently) and transmission errors will occur. Talking in networking terms, it suffices that this hardware

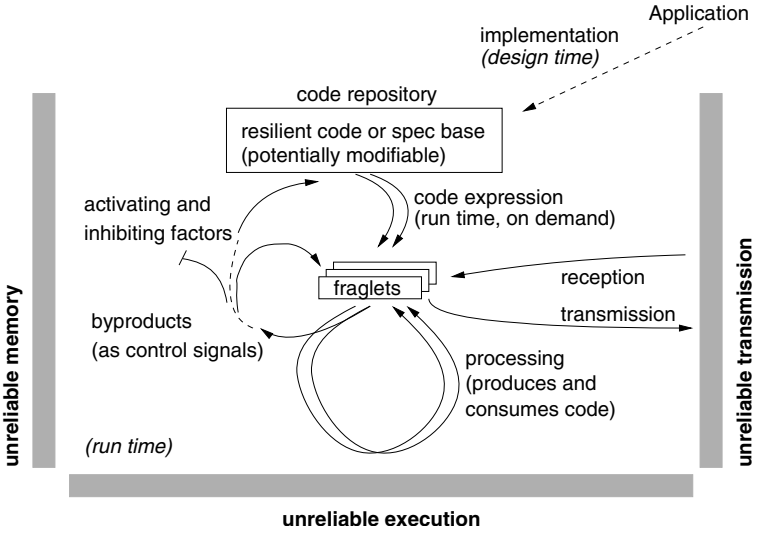


Fig. 1. Execution model for resilient and self-modifying software

offers *best effort services*, memory as well as execution wise. The software is organized in parallel activities, shown as separate execution circles. Activities are basically execution circles because we are working with continuously operating autonomous software.

One hypothesis in our model is that "code" is not permanently installed but is explicitly generated when it is needed. Moreover, Fraglet code is consumed (by executing it), which has the advantage that execution of accidentally modified fraglets is not repeated. Special copy processes (dashed line) are responsible for procuring the code when it is needed[2].

A resilient code repository, shown as a box in Figure 1, could be implemented as specially hardened read-only memory. However, due to our interest in online software evolution, we envisage that the repository is implemented with Fraglets, too: One or more continuous processes watch over the code's integrity and interact with the code expression requests from the copy processes.

The challenging part of such an execution model is the design of the *code generation control loop* that regulates code expression. Fraglet execution should be shadowed by signals (in form of Fraglets) which permit to assess its success or failure. A lost Fraglet execution would be compensated by the recreation of the corresponding Fraglet instance. But also excess executions due to accidentally or deliberately interfering processes (bugs or viruses) could be handled by lowering

[2] When comparing this model to the SWIFT approach (or any classical von Neumann CPU, see the discussion in Section 2 on soft error resilience with SWIFT), we note the similarity in the code copy operation which happens between the memory and the CPU: in SWIFT the code is copied instruction wise. In classical CPUs the code copy process (instruction fetch) is hardwired and cannot be altered.

the rate at which Fraglets are expressed. Thus, we believe that for continuously running software we have to develop code regulation techniques for highly interdependent computations such that all code related errors can be handled in a much more elastic way than the current run-or-break property of sequential programs.

The third research challenge relates to the safe exploitation of the available code self-modification mechanisms of such an execution environment. Ideally, one wants to let a system evolve through its internal code modification procedures while still being assured about long term invariants. Research in programming languages has produced "tamed" versions of self-modification e.g., in form of reflection or the lambda calculus. However, unreliable execution as well as code mobility are not accounted for. We are also not aware of a theoretical framework for code transformations that target self-modifying code and which could, for example, capture self-fragmentation or the dynamics of code regulation.

6 Conclusions

Autonomous software which runs unattended and perpetually should be able to adapt to unforeseen situations and to cope with drastic error conditions at all levels, including external memory and execution faults as well as buggy software components. In this paper we pointed to the properties of adaptability and resilience that such a system must have and opt for a fully software based solution. We argued that important techniques for implementing it boil down to mastering self-modifying code, for which we presented a framework and research roadmap.

References

1. Dewdney, A.K.: Recreational Mathematics – Core Wars (May 1984) Scientific American. See also `http://www.koth.org/`.
2. von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata Studies (1956) 43–98
3. Truszkowski, W., Hallock, H.: Agent Technology from a NASA Perspective. In: Proceedings of the 3rd International Workshop on Cooperative Information Agents (CIA'99). Lecture Notes in Artificial Intelligence (LNAI 1652), Springer-Verlag (1999) 1–33
4. Robertson, P., Williams, B.: A Model-Based System Supporting Automatic Self-Regeneration of Critical Software. In: IFIP/IEEE International Workshop on Self-Managed Systems and Services (SelfMan 2005), Nice, France (2005)
5. Reis, G.A., Chang, J., Vachharajani, N., Rangan, R., August, D.I.: SWIFT: Software implemented fault tolerance. In: Proceedings of the Third International Symposium on Code Generation and Optimization. (2005)
6. Holland, J.: Adaptation in Natural and Artificial Systems. MIT Press (1992)
7. Yamamoto, L., Tschudin, C.: Genetic Evolution of Protocol Implementations and Configurations. In: IFIP/IEEE International Workshop on Self-Managed Systems and Services (SelfMan 2005), Nice, France (2005)

8. Manna, Z., Waldinger, R.: Fundamentals of Deductive Program Synthesis. IEEE Transactions on Software Engineering **18**(8) (1992) 674 – 704
9. Probert, R.L., Saleh, K.: Synthesis of Communication Protocols: Survey and Assessment. IEEE Transactions on Computers **40**(4) (1991) 468 – 476
10. Spector, L., Robinson, A.: Multi-type, Self-adaptive Genetic Programming as an Agent Creation Tool. In: Proceedings of the Workshop on Evolutionary Computation for Multi-Agent Systems (ECOMAS-2002), International Society for Genetic and Evolutionary Computation. (2002)
11. Wetherall, D., Guttag, J., Tennenhouse, D.: ANTS: Network Services Without the Red Tape. IEEE Computer (1999) 42–48
12. Hicks, M., Kakkar, P., Moore, J.T., Gunter, C.A., Nettles, S.: PLAN: A Packet Language for Active Networks. In: Proceedings of the 3rd. ACM SIGPLAN International Conference on Functional Programming Languages (ICFP'98). (1998) 86–93
13. Tschudin, C.: The Messenger Environment MØ - A Condensed Description. In: Mobile Object Systems - Towards the Programmable Internet (MOS'96). Springer LNCS 1222, Linz, Austria (1996) 149–156
14. Tschudin, C.: Fraglets - a Metabolistic Execution Model for Communication Protocols. In: Proceeding of 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS), Menlo Park, USA (2003)

Oracle: An Agent-Based, Reference Architecture

Henry Hexmoor[1] and Jody Little[2]

[1] Computer Science and Computer Engineering Department
Engineering Hall, Room 340A
University of Arkansas
Fayetteville, AR 72701
[2] Sierra Nevada Corporation
4801 NW Loop, 410
San Antonio, Tx 78205
hexmoor@uark.edu, jody.little@sncorp.com

Abstract. This paper introduces a reference architecture for developing agent-based systems that preserves core concepts of agenthood while minimizing cumbersome features found in other agent architectures. This parsimony has been found useful in addressing complex problems.

1 Introduction

With the proliferation of agent-based systems in every type of complex, mission critical system there is a need for a reference architecture that capitulates essential and salient ingredients of agent-based systems. This reference architecture can be used as a guide to organization and design of specific agent-based systems (Gazi, et. al., 2001). In this paper we primarily focus on the salient components and interfaces of such an architecture as we aim to offer a reference architecture in the spirit of James Albus' RCS. A reference architecture is not a blow by blow specification but rather an educated guide for conceptualization. Specific methodologies and operational notions that elaborate our reference architecture will vary among classes of problems. To illustrate this, we briefly outline two case studies in the latter part of this paper.

We take an Agent oriented software engineering approach in exposition of our reference architecture. Agent-oriented software engineering is a proper subset of software engineering (Wooldridge, 2000) with emphasis placed on interactions and relationships among entities we'll call agents. To make explicit agents we offer five guiding principles. First, agents are entities in the problem domain as opposed to models of functional abstractions. More specifically, agents encapsulate computational units that determine plans and actions as well as the process of exhibiting acting. Second, agents are properly sized so they model entities that are rather modest in mass and time. We are not suggesting to model midgets. Rather, we are arguing to consider acting units in such a way that the system being modeled will map to a finite number of agents in order to allow us to meaningfully focus on modeling interesting interactions and relationships. If the agent granules are fairly large we would not have the opportunity to examine intricacies of interagent interface. Since an aim in designing agent-based systems is distributed intentionality, our third principle is that

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 205 – 209, 2006.

agents should be considered to own their local intentionality. System intentionality should be properly divided into a series of proper sets so they can be mapped to intentionality of agent communities and the smallest units map to individual agent intentionality. Our fourth agent design principle is coherent dissemination of information, knowledge, and wisdom. Agents should be provided with methods for caching and sharing results of individually (i.e., locally) processed and fused data. Properly designed dissemination methods will offer cohesion so that the set of agents will act as a whole, i.e., a hive-mind. Although it is beyond the scope of this article, we need to point out the need for shared or disparate ontologies among groups of agents (NCOR). A more elaborated consideration needs to account for delineate ontology mediation by agents themselves or designated agents. Our fifth principle suggests to consider agents operating in sufficient independence as if they operate in parallel. Despite the obvious need for interdepedendence among agents in any complex system and hence models of relationships and interactions suggested herein, agents need to be designed to operate concurrently as opposed to sequentially. The large number of agent architectures in existence does not imply maturity in the discipline. Instead, it reflects a rush to capture and document features of interest. In the following section we will review agent architectures and propose a need for parsimony and a return to original conceptions of modeling agents.

2 Agent Architectures

One of the earliest and the most influential agent architectures is the BDI paradigm. The Stanford group of researchers in mid-1980s suggested capturing mentalistic notions such as belief, desire, intention (Bratman, 1987, Muller, 1997, Rao and Georgeff, 1998). There was a heavy leaning toward grounding BDI in formal modal logics partly to inherit the properties of soundness and completeness and partly to gain expressive power of treating BDI as modalities. The expressive power gained came at the expense of lack of tractability. Along with many researchers we have implemented a limited form of BDI in our labs with partial satisfaction. The best known implementation is often attributed to Kinney and Georgeff, 1991). BDI shortcomings are well-documented and we will avoid repeating them here. Instead, we point to the need to preserve Bratman's claim that rational agents strive to adopt and maintain

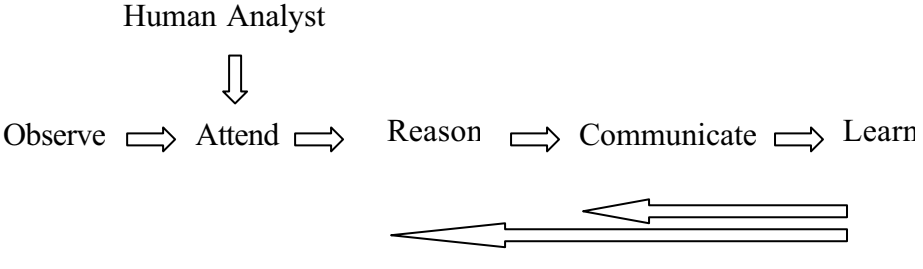


Fig. 1. OARCL Components

conflict-free intentions. All rational agent reasoning will service for avoid detraction from adopted intentions and on methods for manifesting desired objects of intention. We wish to explicate the primacy of *intention* with the need for *attention* as we will see in the following section. A more pragmatic agen t architecture is MaSE (DeLoach, et.al., 2003). Rooted in BDI, DeLoach has not only provided expressive power of modeling roles and communication in MaSE but also provided a blow by blow methodology that was lacking in BDI. Tropos is a recent agent architecture that offers both a methodology and rich expressivity (Giorgini, et.al., 2004).

3 OARCL Reference Architecture

OARCL is comprised of Observe, Attend, Reason, Communicate, and Learn (OARCL) components. Similar to the OODA loop concept proposed by Col. John Boyd (Richards, 2004), it is complex and involves many internal loops and nontrivial processes that supervise and keeps the system inline with various measures of effectiveness and performance. There are abundant asynchronies and nonlinearities (see Figure 1). One of our aims in introducing Oracle is to urge a return to the original conceptions of agents, which were anthropomorphic modeling of a sense of acting. Another aim is to put in the spotlight the salient properties of agency by Oracle components. At a metaphorical level, an OARCL agent is a cognitive entity with functionalities suggested by pre-attentive functions captured in the module Observe, attention generation and maintenance in the module Attend, inferential capabilities in the component Reason, intentional message generation in the module Communicate, and abilities to modify perception, attention, and reasoning processes in the component we call Learn. Next we provide general description for each component.

In order to coarsely filter out irrelevant data, the Observe component performs agent input data distribution management functions. Agent who live in a highly distributed environments need to control the volume and the nature of input they process. We choose to consider the tasks of what kind and how much data to process as a pre-attentive function of an agent. The best example is human visual processing where visual input is rapidly and automatically filtered. Supported by recent findings in human visual processing that suggests *attention* plays an important cueing role even in early, *preattentive* visual stage we have selected that as our next component (Healey, et.al,.1993) .

Attend receives references from the human supervisor, all agent requests from the Observe component as well as from the Communicate module. References might be associated with specific sensory-identified objects or targets or it might be in terms of features, patterns, and conditions. The selection process, which includes nontrivial reasoning is modeled in this component. We will deliberately leave out discussion of the obvious connection between attention to awareness and consciousness (Crick, 1996). Despite this omission, we point out that this connection is the most elusive, intriguing, yet essential character of agency. There is an inextricable relationship between defining characteristics of independence and pro-activity of computational agents and self-awareness encapsulated in attention. The notion of individuality in attention plays a crucial role in guiding as well as controlling inference and logical reasoning. The rationality principles of Allen Newell and Nick Jennings are addressed

in our reason module (Boman and Van deVelde, 1997). The *reason* component performs the primary inference in OARCL agents.

Communicate performs external information management including the speech act using standard agent communications language (ACL) for outputs (e.g., requests) and input information and requests from other agents.

Finally, the Learn component generates performance assessment that drives its process management of all agent components. Next, we will discuss applications that lead to development of our reference architecture.

One of the problems that motivated Oracle was the SIGINT man on the loop with the aim to design an agent-based software that aids and automates intelligence analysis, technically known as SIGINT analysis (Hollywood, et.al, 2004). The SIGINT activity encompasses all of command, control, communications, human intelligence, surveillance, and reconnaissance. The second motivating problem was modern disaster response with the aim to design an agent-based software that fuses information at varying levels of abstraction in order to rapidly assess situations at a high level. Our approach preserved the highly distributed and disparate loci of information gathering and synthesis.

Next, we review Oracle modules and briefly discuss the software engineering tasks therein. For Observe one must broadly gather techniques for selecting data sources and channels with details beyond the scope and interest of this paper. Attend for both problems must be flexible to allow for changes in the human analyst's goals and targets. Generically, a human analyst will set and revise conditions and targets of interest for the remainder of the system (shown in Figure 1). The reference points selected in attend will be used to place filters on data gathered by Observe. Reason will need to identify threats and opportunities of interest n the command and control, which is the core function of a typical human analyst.

Communicate will consist of (a) all conditions for triggering messages to other system functions in support of SIGINT as well as disaster response, and (b) types and formats of messages corresponding to triggering conditions.

Finally, Learn will embody metrics and sets of adjustments for internal functions as well as interactions among Oracle modules.

4 Conclusion

Oracle is both a call to return to original conceptions of agenthood and agent-based research as well as a reference architecture for designing agents in an agent-based system. Our outline here is very concise and does not offer a detailed methodology. It also does not provide properties found in aspect –oriented paradigm.

Acknowledgements

Prof. Henry Hexmoor wishes to acknowledge support and encouragement of staff at Sierra Nevada Corporation's San Antonio Operations as well as the Center for Multisource Fusion at SUNY Buffalo.

References

1. M. Boman, M. and Van de Velde, W. Multi-Agent Rationality: Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 13-16, 1997.
2. Bratman.M.E., *Intentions, Plans, and Practical Reason*. Harvard University Press: Cambridge, MA, 1987.
3. Crick, F. Visual perception: rivalry and consciousness. *Nature* 379:485-486, 1996
4. DeLoach, S.A., Matson, E.T., Li, Y.. Exploiting Agent Oriented Software Engineering in the Design of a Cooperative Robotics Search and Rescue System**.** The International Journal of Pattern Recognition and Artificial Intelligence, 17 (5), pp. 817-835, 2003.
5. Hollywood, J., Snyder, D., McKay, K. N., and Boon, J.E. Out of the Ordinary: Finding Hidden Threats by Analyzing Unusual Behavior, Rand pub, ISBN: 0-8330-3520-7, 2004.
6. Gazi, V., Moore, M.L., , Passino, K.M., Shackleford, W.P., Proctor, F., Albus, J.S. The RCS Handbook: Tools for Real Time Control Systems Software Development, Wiley, 2001.
7. Giorgini, P., Kolp, M., Mylopoulos, J., and Pistore, M. The Tropos Methodology: an overview. In F. Bergenti, M.-P. Gleizes and F. Zambonelli (Eds) Methodologies And Software Engineering For Agent Systems, Kluwer Academic Publishing, 2004.
8. Healey, C. G., Booth, K. S., and Enns, J. T. Harnessing preattentive processes for multivariate data visualization. *Proceedings Graphics Interface '93* (Toronto, Canada, 1993), pp. 107–117, 1993.
9. Kinny, D. and M. Georgeff., M. Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 82–88, Sydney, Australia, 1991.
10. Muller, J.P. *The Design of Intelligent Agents (LNAI Volume 1177)*. Springer-Verlag: Berlin, Germany, 1997.
11. Rao, A.S. and Georgeff, M. Decision procedures of BDI logics. *Journal of Logic and Computation*, 8(3):293–344, 1998.
12. Richards, C. Certain To Win: The Strategy Of John Boyd, Applied To Business, Xlibris Corporation publisher, 2004.

Hierarchies, Holons, and Agent Coordination*

Albert Esterline[1], Chafic BouSaba[2], Barbara Pioro[1], and Abdollah Homaifar[2]

[1] Department of Computer Science
[2] Department of Electrical and Computer Engineering
North Carolina A&T State University, Greensboro, NC
esterlin@ncat.edu, pioro@ncat.edu, cbousaba@ncat.edu, homaifar@ncat.edu

Abstract. We address tactical behaviors of unmanned ground vehicles (UGVs) and focus on their coordination within hierarchical units. This research thus addresses multiagent systems but more particularly structures where autonomous parts coordinate to form wholes that in turn are parts in more inclusive wholes. To capture these notions, we introduce the concept of a holarchy (a hierarchy of holons, which are both parts and wholes) and use an enhancement of the Statechart notation called Parts/whole Statecharts. We relate aspects of this notation to the technical notion of common knowledge, which is a necessary condition for coordination, and we sketch techniques for building holarchic models using Parts/whole Statecharts. An implementation of one such model in Matlab's Stateflow tool is sketched.

1 Introduction

With increased use of military unmanned vehicles, there is a critical need to model the behaviors of units of unmanned vehicles to facilitate development of coordination protocols and tactical policies that are effective in adverse situations. We address tactical behaviors [1] of unmanned ground vehicles (UGVs) and focus on their coordination within hierarchical units. This research thus addresses multiagent systems but more particularly structures where autonomous parts coordinate to form wholes that in turn are parts in more inclusive wholes.

In the next section, we introduce the notion of a holarchy, which captures the required idea of parts and wholes. We also present a military reference model architecture based on this notion and an outline of tactical behaviors. And we introduce the enhancement of the Statechart notation (called Parts/whole Statecharts) that we use for representing holarchies formally; we relate aspects of this notation to the technical notion of common knowledge, which is a necessary condition for coordination. In section 3, we discuss a model implemented in Matlab's Stateflow tool that uses the conventions of Parts/whole Statecharts to capture the holarchic structure of a UGV section. In the next section, we sketch techniques for building holarchic models using the Parts/whole Statechart notation. Section 5 concludes and discusses future work.

* This work is supported by General Dynamics Robotics System, grant "Learning and Adaptation for Tactical Behavior". Guidance from Mr. D. Rogers of General Dynamics is gratefully acknowledged.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 210–221, 2006.

2 Background

We here introduce the notion of a holarchy and present a military reference model architecture based on this notion and an outline of tactical behaviors. We also introduce Parts/whole Statecharts, which we use for representing holarchies formally; we relate aspects of this notation to the technical notion of common knowledge.

2.1 Multiagent Systems and Holarchies

Wooldridge defines an agent as a computer system capable of autonomous action that meets its design objective in the environment in which it is situated [2]. An agent is autonomous in the sense that it has a say in whether and how it responds to a request. Multiagent environments, according to Huhns and Stevens, "provide an infrastructure specifying communication and interaction protocols [3]," and coordination is critical in the functioning of a multiagent system. Complex coordinated operations, however, require social organization about which the general multiagent-system paradigm is mute. Social institutions with efficient communication and control have hierarchical structures. Simon [4] has noted how hierarchy defeats complexity not only in terms of our understanding of systems but also in terms of the stability of the systems themselves. If, however, an institution is to succeed in novel situations, the components in its hierarchy, all the way down to the individuals with no command responsibilities, must be given the initiative within certain bounds.

To analyze coordination hierarchies of agents, we use Koestler's notion of a holarchy [5], which is a hierarchy of holons. A holon is both a whole (self-assertive) and a part (cooperative). "Holon" is a combination of the Greek word "holos," meaning whole, and the suffix "on," meaning particle or part. Koestler viewed holons as filling roles and as defined by fixed rules and flexible strategies, as best illustrated perhaps by language, which has fixed grammatical rules yet supports endless forms of conversation. In engineering domains, holarchies appear mostly in the form of holonic manufacturing system (HMSs) – see, e.g., [6,7] – which provide ways to organize manufacturing systems so that key elements (i.e., holons), such as machines, plants, products, personnel, and departments, have autonomous and cooperative properties. A HMS is highly agile because the holons coordinate their activities with each other and are not centrally controlled. Problems extensively addressed in the literature include defining holons for a given problem and designing efficient communication and coordination protocols.

2.2 4D/RCS Reference Model and Tactical Behaviors

The 4D/RCS architecture [8] is a complete reference model for UGVs consistent with military hierarchical command doctrine. A generic control node, called an RCS_NODE, is the building block for all the hierarchical control levels and enables intelligent behavior. Figure 1 is a block diagram of a 4D/RCS reference model for a notional Future Combat System (FCS) battalion. Each RCS_NODE

is an identifiable part of the 4D/RCS system that has a unique identity yet is made up of sub-ordinate parts and in turn is part of a larger whole. The RCS_NODE satisfies Koestler's definition of a holon [5]. In particular, each RCS_NODE is both self-reliant and cooperative.

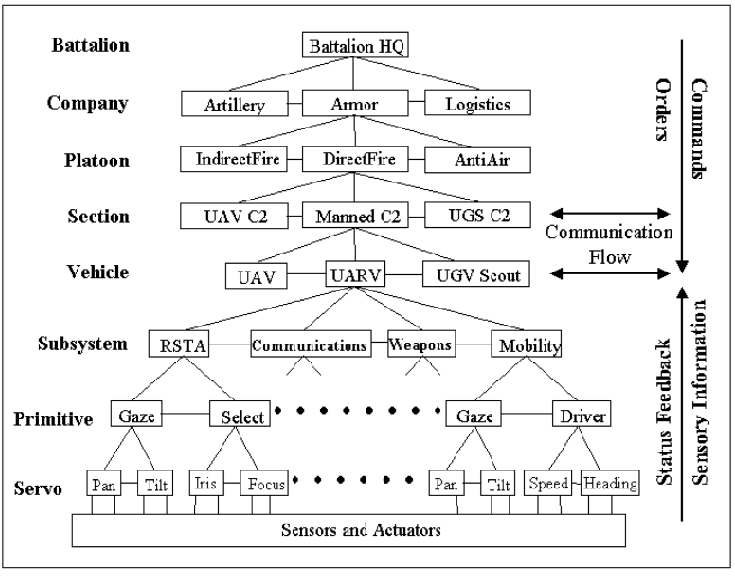


Fig. 1. A 4D/RCS Model for a Battalion

According to the U.S. Army [1], a movement formation is the configuration of the vehicles in a unit while a movement technique is a method used by units to traverse terrain. In our domain, a UGV may assume one of two roles, Lead or Follower. A section assumes one of three movement formations: Column (one after the other), Staggered Column (the same but staggered), or Line (one beside the other). And it assumes one of three movement techniques: Traveling (fast with little security), Traveling Overwatch (where the trail covers the lead, giving greater security with less speed), or Bounding Overwatch (the most secure and slowest).

2.3 Parts/Whole Statecharts

A holarchy is a concurrent structure, and the most popular automata for modeling concurrent systems are Statecharts [9], a visual formalism for the behavioral description of complex systems that extends classical state diagrams in several ways. A superstate that contains substates and transitions that elaborate its sequential behavior is called an XOR state since, when it is active, exactly one of its substates is active. A superstate that contains concurrent substates –"orthogonal

components"– is called an AND state. A basic state has no substates so is neither an XOR nor an AND state. A transition has a label $e[c]/a$ indicating that, when the system is in the source state, condition c holds, and event e happens, then it can move to the target state on performing action a; the condition and action are optional. A transition may have a non-basic state as either its source or target. Frequently, the condition indicates that an orthogonal component is in a given state. Finally, a history pseudostate (denoted by H) may be included in a state. It indicates that, when the state is re-entered, the substate that was last active should again become active. The hierarchical structure of a Statechart makes it easy to suppress detail (zoom out) or to focus on detail (zoom in).

For example, consider the Statechart in Figure 2. The overall state, s, is an AND state: the system must be in both its orthogonal substates, u and v. States u and v are XOR states. In state u, the system must be in either m or k (but not both). Similarly for states m (sub-states $n1 - 2$) and v (substates $p1 - 4$). An arrow starting from a darkened circle points to the default initial substate of an XOR state. The label on the transition from k to m indicates that, in state k, if event $a1$ happens, the system can move to state m on performing action $b1$. Likewise for the other transitions. An action done by one orthogonal component can be a triggering event for another. E.g., when component u transitions from k to m, it performs action $b1$, which is the triggering event for component v when in state $p1$ (which may then transition to state $p2$).

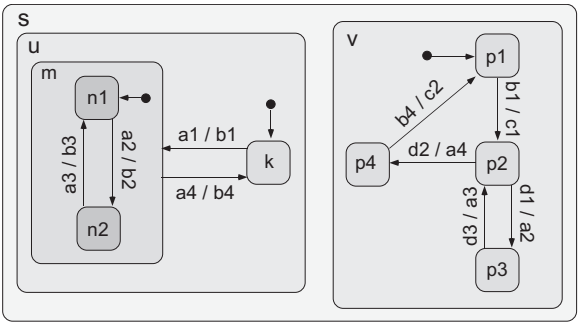


Fig. 2. Statechart Example

The Statechart formalism is characterized not only by hierarchy (XOR states) and orthogonality (AND states) but also by broadcast communication: an action in one orthogonal component can serve as a triggering event in other components. This is one way components of a Statechart are implicitly coordinated. Other ways are by transition conditions that test the state of another component and by using the same event as a trigger in transitions in different orthogonal components. This implicit coordination introduces a web of references and eliminates encapsulation. We therefore follow Pazzi [10] in using an explicit approach to modeling aggregate entities, introducing a whole as an orthogonal component on a par with the parts that are coordinated. A part communicates only with the

whole, never with other parts. As an example, the Statechart in Figure 2 can be converted to a Parts/whole Statechart by adding (to the AND state s) the whole shown in Figure 3 and relabeling events $a2$, $a3$, and $a4$ in u as $wa2$, $wa3$, and $wa4$ and events $b1$ and $b4$ in v as $wb1$ and $wb4$. Statecharts usually do not involve the decomposition of aggregate entities to more than one level, but with Parts/whole Statecharts this becomes natural. A parts/whole Statechart thus supports two kinds of hierarchies, XOR hierarchies and parts/whole hierarchies, the latter modeling holarchies.

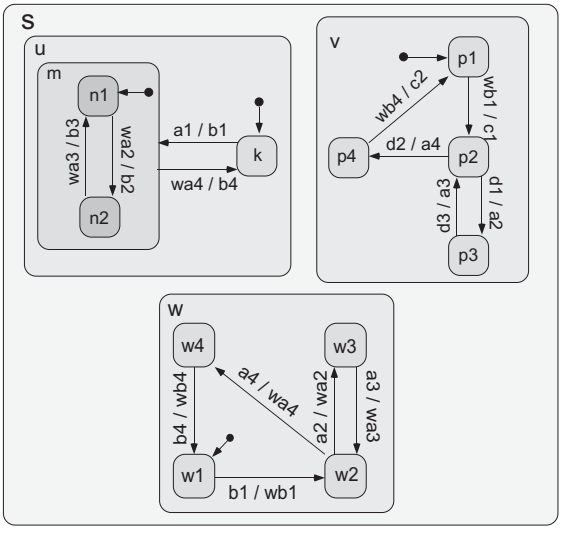


Fig. 3. Constructing a whole

2.4 Common Knowledge

Since the purpose of the whole is coordination, we relate the whole in a team or society to behavior that rests on common knowledge, which is a necessary condition for coordination [11]. The notion of common knowledge was introduced in the 1960s by the philosopher David Lewis. In game theory, games of complete information assume that players' strategies and payoffs are common knowledge, while games of perfect information assume that players' moves are common knowledge. Common knowledge has also been used in the analysis of protocols in distributed systems [11]. Recent work in psycholinguistics [12] emphasizes coordination, hence common knowledge, in conversation.

Where ϕ is a proposition and G is a group, it is common knowledge in G that ϕ if everyone in G knows that ϕ, everyone in G knows that everyone in G knows that ϕ, and so on for arbitrarily deep nestings of the operator "everyone in G knows that." For example, for traffic lights to serve their purpose, it is not enough for everyone to know that red means stop and green means go. It must also be the case that everyone knows that everyone knows this. Otherwise, for

example, no one could enter an intersection with confidence on a green light. The notion of common knowledge can be analyzed more perspicuously in terms of a fixed point. Let $E_G\phi$ mean that everyone in group G knows that ϕ, and let $C_G\phi$ mean that it is common knowledge in G that ϕ. Then we can define $C_G\phi$ as the solution to the logical equivalence $C_G\phi \iff E_G(\phi \wedge G_G\phi)$, which characterizes $C_G\phi$ as a fixed point. We can also characterize common knowledge (following Clark and Carlson [13]) in terms of shared situations, which in some sense embody this fixed-point equivalence. Assume that A and B are rational agents. Then we may infer common knowledge among A and B that ϕ if (1) A and B know that some situation σ holds, (2) σ indicates to both A and B that both A and B know that σ holds, and (3) σ indicates to both A and B that ϕ. Barwise [14] concluded that the fixed-point approach is the correct analysis of common knowledge, but that common knowledge generally arises via shared situations. Indeed, a situation "provides a stage for maintaining common knowledge through the maintenance of a shared situation."

3 A StateFlow Model

We sketch a Parts/whole Statechart modeling the movement formations and techniques of a section of military UGVs. This Statechart was implemented in Stateflow, an interactive Statechart design tool integrated with MATLAB and Simulink. We consider a section formed of two UGVs (the lead and follower). The section's task is divided into two phases. The first –see Figure 4– is a tactical road march (column movement formation and traveling movement technique) until both UGVs reach the checkpoint at the tree line. After synchronizing, phase two, reconnaissance (staggered column movement formation and traveling overwatch movement technique), starts. The task finishes when the UGVs reach their final destination. A section may change its tactical behavior in response to critical events, such as environmental changes and loosing the line of sight between lead and follower. After coping with these situations, the task is continued where it was interrupted.

Figure 5 portrays the top level of the Parts/whole Statechart representing this task. The components of the *Environment* state generate random and critical events, which are detected by the UGVs and result in change of behavior. At every cycle, *Random_Event_Generator* and *Random_Exception_Generator* randomly generate an anticipated event (e.g., finding a tree line), a critical, or unplanned, event (e.g., loss of the line of sight), or no event at all.

Terrain_Classification captures changes in three aspects of the terrain: cultural features, elevation, and vegetation.

The *Section* state represents a parts/whole decomposition. The two parts are *Lead_UGV* (leader role in a UGV section) and *Follow_UGV* (the follower role), and *Section_Whole* is the whole. Any event detected by either UGV is communicated with the other by virtue of the whole. Figure 6 zooms in on *Lead_UGV* and *Section_Whole*. (*Follow_UGV* is somewhat analogous to *Lead_UGV*.) *Speed_L* is an XOR state that qualitatively records the speed of the

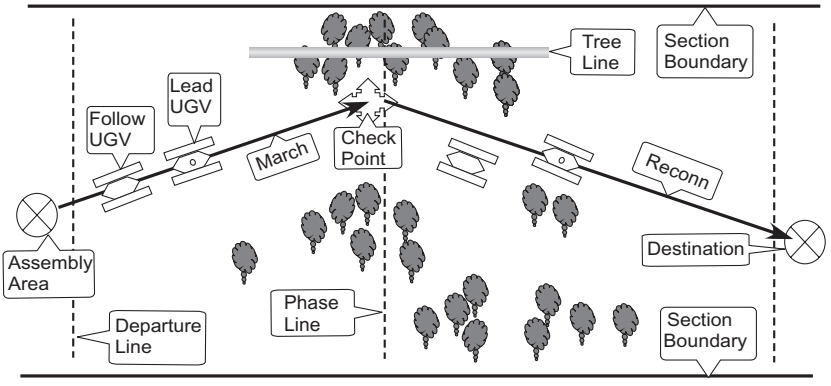


Fig. 4. Section Task

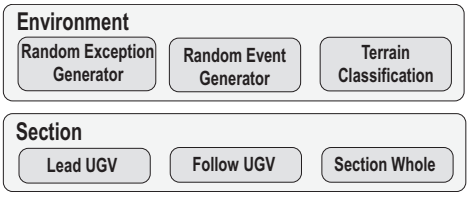


Fig. 5. Model Architecture

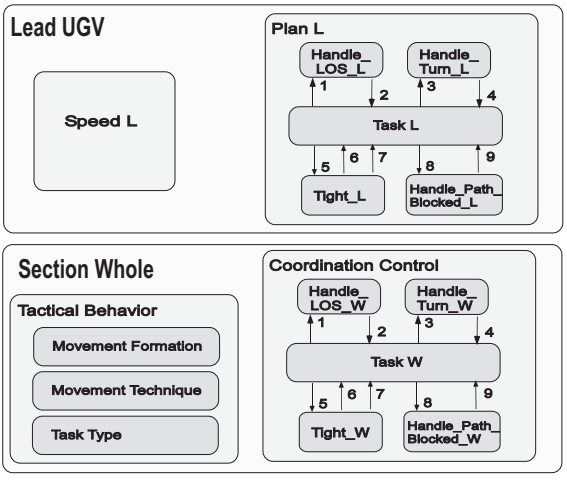


Fig. 6. Lead and Section1 Whole

lead UGV with substates such as *Slow_L*. The substates in *Tactical_Behavior* are XOR states. *Movement_Formation* represents four possible formations, *Movement_Technique* represents four possible techniques, and *Task_Type* has

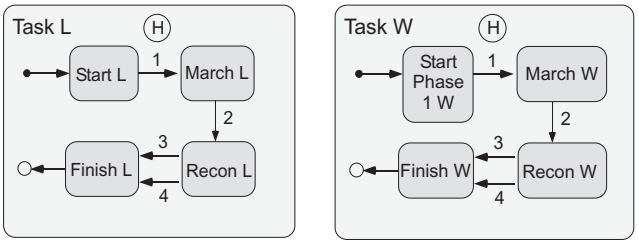


Fig. 7. Lead UGV and Section1 Whole Tasks

Table 1. Key for labeling of transitions in figure 7

Number	Transition Label	Relates to
Task L		
1	after(1,Go)/Coord....L_Go_W	Start marching
2	L_Line_Found/Coord....L_Line_W	Lead finds line
3	W_Region_L/	Follower finds destination
4	L_Region_Found/Coord....L_Region_W	Lead finds destination
Task W		
1	L_Go_W/Follow.W_Go_F	Start marching
2	L_Line_W/Follow.W_Line_F	Lead finds line
3	F_Region_W/Lead.W_Region_F	Follower finds destination
4	L_Region_W/Follow.W_Region_F	Lead finds destination

substates for three types of task (*Marching*, *Reconn*, and *Engaging*). *Plan_L* (in *Lead_UGV*) and *Coordination_Control* (in *Section_Whole*) have parallel structure (along with has *Plan_F* in *Follow_UGV*) since we represent only events that impact coordination. A more detailed model would show considerably more structure for the UGV parts to indicate behavior that need not be coordinated. *Task_L* represents the main path of play for the lead, and *Task_W* is the corresponding synchronization skeleton. Adaptation to four kinds of critical events is indicated: loss of line of sight and detection of a turn in the path, a narrow path, and a blocked path. The transitions are numbered in *Plan_L* and *Coordination_Control* so that transitions with the same number interact. History pseudostates in *Task_L* and *Task_W* ensure that, when the response to a critical event is finished, the main path of play is taken up where it left off when the event occurred.

Figure 7 zooms in on substates *Task_L* and *Task_W*, which capture the anticipated phases of the task, marching and reconnaissance. The transitions, labeled with numbers, are explained in Table 1. By convention, the names of coordination

events and actions are of the form X_Name_Y, where X is L, F, or W depending on whether the action is performed by the lead, follower, or whole, respectively; Y has the same possible values and indicates the part for which the action serves as a triggering event. Each action is prefixed with the name (followed by a dot) of the component (e.g., $Section\_Whole$) where it serves as a triggering event if that component is not the current component. Space restrictions prohibit further detail on this model, which contains over 200 states.

4 Constructing Parts/Whole Statecharts

We here outline a few heuristic techniques for constructing a Parts/whole Statechart from an arbitrary Statechart and building a Parts/whole Statechart holarchy. Rigorous procedures would probably rely on a process algebraic-semantics for Statecharts, such as that presented in [15]. In fact, some of the terminology we introduce here has a process-algebraic inspiration.

4.1 Introducing a Whole

To begin with, consider the case where there are two coordinating parts. We first identify cases where the action on the transition in one part (called the *actor*) is the triggering event on the transition in the other part (called the *target*). We call such action-event pairs *complementary pairs*, and we call the triggering of the transition in the target by the performance of the action in the actor a *handshake*. There is a transition in the whole for every allowable handshake. This requires one to rename either the action (in the actor) or the event (in the target); we assume that the event is renamed. Then the corresponding transition in the whole has the actor's action as its event and the target's event as its action. Since the same complementary pair may occur in several ways, the construction need not be unique. Handshakes may cascade, with an action in one component triggering a transition hence an action in another component, which may trigger an action in a third component, and so on. Ignoring non-coordination events and actions, one generally tries to reflect as much of the XOR hierarchical structure of the coordinated parts as possible. A state referenced in conditions in other parts should be duplicated in the whole. References to the original state in conditions are replaced with references to its image in the whole. This requires much of the structure of the part with the referenced state to be duplicated in the whole. If one part is itself an AND state without a coordinating whole, it is replaced by its substates. Cases where there are more than two coordinating parts are handled much like cases with only two. Cascades of handshakes may involve more than two parts, but this presents no difficulty.

Now, a given action in one part may have complimentary events in several other parts. This allows two kinds of coordination, parallel and serial. For parallel coordination, the events in the targets are all renamed with the same name, viz., the complementary action in the whole. This allows multiple concurrent handshakes. For serial coordination, the targets are ordered, and the event in the first target becomes the compliment of the action in the whole transition that

handshakes with the action in the actor. We introduce a subsequent transition in the whole that handshakes (via its event) with the transition in the first target and (via its action) with the second target. Further transitions in the whole and renaming of the events in question in the remaining targets support a cascade of handshakes. When parts are coordinated serially, we often must allow several orders in the cascade to be equivalent, so we need a path through the whole for each order. For parallel coordination, we would generally want a distinguished part that can perform actions whose complements are events in the shared interfaces of the other parts. For sequential coordination, we must decide on an order in which handshakes are cascaded, which requires ranking the parts. We can think of the distinguished part in the case of parallel coordination or the highest ranked part in serial coordination as the leader.

4.2 Forming a Parts/Whole Hierarchy

We can obtain a parts/whole hierarchy of arbitrary height by using the techniques outlined above for forming a whole except that, once we move up from the lowest hierarchy level, wholes are introduced to coordinate other wholes. A whole may coordinate a mixture of parts and wholes, and, in general, there is no need to partition the hierarchy into levels.

Now, coordination between groups might be such that the action patterns involved in the coordination are common knowledge among the members of both groups. For larger or more structured groups, however, it is not reasonable to expect each member to be as knowledgeable as this would require. For an alternative view, consider distributed knowledge [11]. Group G has distributed knowledge that ϕ if the members of G can pool there knowledge and infer that ϕ. Now, it is often the case that, for groups G and H to coordinate, it suffices that group G has the distributed knowledge required for the coordination and that group H likewise has this distributed knowledge. That is, there is common knowledge in the group consisting of G and H but not necessarily in the group consisting of the members of G and the members of H. A particularly simple case is where the relevant distributed knowledge of group G is known by a single member and likewise for the distributed knowledge of H. Then the common knowledge needed for coordinating the groups is in fact the common knowledge not only of these groups but also of these distinguished members, which we might think of as representatives. A group may have a leader in the sense mentioned above and a representative; it need not be the same member that fills both roles. Many other kinds of structure are possible with parts/whole hierarchies.

5 Conclusion and Future Work

We have shown how to model the tactical behaviors of hierarchical units of military UGVs as holarchies, that is, hierarchies of holons, where a holon is both a whole and a part. Holarchies are represented with Parts/whole Statecharts, which introduce a whole in an AND state so that the parts (given orthogonal components) interact with the whole. The whole thus coordinates the parts,

and we relate it to the technical notion of common knowledge, which is a necessary condition for coordination. As an example, we presented a Parts/whole Statechart, implemented in Matlab's Stateflow tool, modeling the behavior of a section of two UGVs performing a task. Finally, heuristic techniques were sketched for introducing wholes into Statecharts and for building parts/whole hierarchies that model holarchies.

Space restrictions preclude a discussion of how a holarchical Parts/whole Statechart model may be implemented by instantiating parts in individual agents. The major issue here arises from the fact that a whole does not correspond to an individual in the system modeled. The main insight is that a whole is instantiated in all the individuals that instantiate parts coordinated by the whole, but each individual instantiates the whole from its own point of view: details not affecting that part may be abstracted away.

Holarchic modeling with Parts/whole Statecharts is seen to be a powerful modeling technique that is intuitive and that facilitates modular, hierarchical analysis of complex systems. Our framework shares some important features with general system theory. Koestler's theory of holons is considered to be in the general area of general system theory, and hierarchical structures are often invoked by general system theorist as key to grasping the domain of systems, namely, "organized complexity,"which is too complex for analysis and too organized for statistics [16]. Von Bertalanffy maintained that "General system theory is a general science of ' wholeness' [2]." Weinberg's discussion of observers in [16] culminates with the notion of a superobserver (relative to a group of observers) whose observations are similar to the distributed knowledge of the group.

In the future, we intend to adapt insights from general system theory to our formalism. We also intend to develop graphical interfaces and environments for our Stateflow models to facilitate broader experience with more involved holarchies. Finally, we shall seek formal support for our concepts both by applying process-algebraic semantics to our Statecharts and by using the HiVy [17] tool developed by the Jet Propulsion Laboratories for translating Stateflow Statecharts into the design language of the SPIN model checker.

References

1. U.S.Army: Field Manual 3-19. Headquarters, Department of the Army, Washington, D.C. (1993)
2. Wooldridge, M.: Intelligent Agents. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT press, Cambridge, MA (2000) 27–78
3. Huhns, M., Stephens, L.: Multiagent Systems and Societies of Agents. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (2000) 79–120
4. Simon, H.A.: The architecture of complexity: Hierarchic systems. In: Proc. of the American Philosophical Soc. Volume 106. (1962) 467–482
5. Koestler, A.: The Ghost in the Machine. Macmillan, New York (1968)

6. K. Fischer, M.S., Siekmann, J.: Holonic multiagent systems: The foundation for the organization of multiagent systems. In V. Marík, D.C.M., Valckenaers, P., eds.: HoloMAST. Volume 2744 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2003) 71–80
7. Giret, A.: A multi agent methodology for holonic manufacturing systems. In: AAMAS '05: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, ACM Press (2005) 1375–1375
8. Albus, J.: 4D/RCS: A reference model architecture for unmanned vehicle systems. In: Proceedings of the SPIE 16th Annual International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Orlando, FL, NIST (2002)
9. Harel, D.: Statecharts: A visual formalism for complex systems. In: Science of Computer Programming. Volume 8. (1987) 231–274
10. Pazzi, L.: Implicit versus explicit characterization of complex entities and events. In: Data and Knowledge Eng. Volume 31. (1999) 115–134
11. R. Fagin, J. Y. Halpern, Y.M., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge, MA (2003)
12. Clark, H.: Using Language. Cambridge University Press, Cambridge (1996)
13. Clark, H., Carlson, T.: Speech Acts and Hearers' Beliefs. In: Mutual Knowledge. Academic Press, London (1982) 1–36
14. Barwise, J.: The Situation in Logic. CLSI Publications, Stanford, CA (1989)
15. Uselton, A.C., Smolka, S.A.: A process algebraic semantics for statecharts via state refinement. In: PROCOMET. (1994) 267–286
16. Weinberg, G.M.: An Introduction to General Systems Thinking. 1st edn. Dorset House Publishing Company, Inc. (2001)
17. Pingree, P.J., Mikk, E.: The HiVy tool set. In R. Alur, D.A.P., ed.: Computer Aided Verification: 16th International Conference, Boston, MA, Springer-Verlag (2004) 466–469

A Systemic Framework for Open Software Agents

Eric Sanchis

Laboratoire de Gestion et Cognition
IUT Ponsan - Université Paul Sabatier
115 route de Narbonne, 31077 Toulouse, France
sanchis@iut-rodez.fr

Abstract. The systemic theory associates open systems and complexity closely. This article presents a particular articulation between these two concepts using the Systemion Model. Two types of opening are defined, characterized and illustrated using examples.

1 Introduction

Computer systems are more and more complex both at the level of their composition and of their behaviour. The complexity of these systems has become so extensive that the most experienced system administrators feel more and more difficulty in ensuring their good functioning. But how can a complex system be identified? According to our point of view, a *complex system* is composed of heterogeneous elements with not clearly defined components and fuzzy interactions, whose total functioning cannot be deducted from the functioning of its elements.

It is important to note that the computer systems do not constitute the only field where complexity poses a major intellectual challenge: complexity is also present in biological, physical and socio-economic systems. For half a century this inevitable complexity has given rise to numerous studies and researches which has led to two very different approaches: (1) equational approach and (2) structural approach.

The *equational approach* is articulated around an extreme simplification of the basic elements of the system, allowing the use of powerful mathematical formalisms the treatment of which has led to an interpretation of the total behaviour of the system [16], [1].

The *structural approach* refutes the reduction of the elements of the system in simple indiscernible items and postulates that it is possible to decompose the system into elements and interactions without denaturing their intrinsic complexity. Contrary to the equational approach, the structural approach does not suppress the internal architecture of the elements of the system. *Multiagent systems*, *autonomic computing* [10] and the various theories resulting from the systemic thought such as the *hierarchical complex systems* [15], the *systemography* [12] or *systemions* [14] belong to the structural approach. Besides, the structural method insists on the architectural characteristics of the elements which compose the complex system. According to the relations which it maintains with its environment, the system will be called *open system* or *closed system*. We shall present two aspects completely different from the opening of

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 222 – 232, 2006.

a system: the *informational opening* and the *physical opening*. It is this last type of opening which we shall illustrate in a more precise way by applying it to the concept of agent. We shall use it to introduce its internal architecture (two-layer architecture) and identify the properties which introduce complexity within the agent (qualities). Linking the physical opening to an architecture with two levels (one layer ensuring the stability of the entity, the other taking care of its internal transformations) seems to be an interesting way of research in the implementation of self-management within complex computer systems.

This paper is structured as follows. Section 2 presents two definitions of the notion of open system, illustrating two different points of view on this concept. The first definition was proposed by Carl Hewitt in order to have a formal model of calculation adapted to the study of distributed systems. The second definition arises from General System Theory and gives itself a vaster field of applications - natural or artificial physical systems to the most complex social organizations. The interest of these two definitions is that each one of them introduces an aspect which is different from the concept of opening. The first is mainly based on the concept of informational opening; the second integrates the exchanges of materials between the system and its environment, i.e. what we will name the physical opening. The section will end with a very general description of a software agent model whose architecture allows the implementation of the physical opening. This model of agent results from one of the components of General System Theory: the systemic approach. The following two sections will introduce the supplementary abstract tools necessary for a precise presentation of an open agent model called the Systemion Model. Finally, the last section will illustrate the realization of the physical opening in this model by using two examples.

2 From Open Systems to Open Agents

The expression *open system* was used in various disciplines, in old ones (physics, biology) as well as new ones (sciences of the systems, data processing). Each one of them has provided this concept of characteristics, for some of them, very general and largely applicable to many contexts, for others very specific to a particular field. Three uses of this expression can be mentioned (1) it is sometimes used to characterize an elementary entity taking part in the more general activity of an organization, (2) in certain contexts, the expression even applies to the organization itself and not to its components, (3) in other fields, the expression is applied at the same time to the component and to the whole. The object of our work relates more to the architecture and the functionalities of the unit of execution than the characteristics of the organization to which it belongs. Consequently, after having specified what the concept of open system means, our matter will concentrate on the *open agent* concept.

2.1 Open Systems

Carl Hewitt defines and characterizes the open systems in the following way: "*Open systems deal with large quantities of diverse information and exploit massive concurrency. They can be characterized by the following fundamental characteristics:*

(1) ***Concurrency.*** *Open systems are composed of numerous components such as workstations, databases, and networks. To handle the simultaneous influx of information from many outside sources, theses components must process information concurrently*

(2) ***Asynchrony.*** *There are two sources of asynchrony in open systems. First since the behavior of the environment is not necessarily predictable by the system itself, new information may enter the system at any time, requiring it to operate asynchronously with the outside world. Second, the components are physically separated distances prohibiting them from acting synchronously. Any attempt to clock all the components synchronously would result in an enormous performance degradation because the clocks would have to be slowed down by orders of magnitude in order to maintain synchronization*

(3) ***Decentralized control.*** *In an open system, a centralized decision maker would become a serious bottleneck. Furthermore, because of communications asynchrony and unreliability, a controlling agent could never have complete, up-to-date information on the state of the system. Therefore control must be distributed throughout the system so that local decisions can be made close to where they are needed*

(4) ***Inconsistent Information.*** *Information from outside the system or even from different parts of the same system may turn out to be inconsistent. Therefore decisions must be made by the components of an open system by considering whatever evidence is currently available*

(5) ***Arm's-length relationships.*** *The components of an open system are at en arm's- length relationship: the internal operation, organization, and state of one computational agent may be unknown and unavailable to another agent for reasons of privacy or outage of communications. Information should be passed by explicit communication between agents to conserve energy and maintain security. This ensures that each component can be kept simple since it only needs to keep track of its own state and its interfaces to other agents*

(6) ***Continuous operation.*** *Open systems must be reliable. They must be designed so that failures of individual components can be accommodated by operating components while the failed components are repaired or replaced*"[6].

A key concept of this definition is the information notion considered as input and output streams. This informational opening is an essential characteristic of Hewitt's open system definition. This point of view is in accordance with computers since their origin, i.e. systems of data processing. The internal architecture of these systems is not at all affected by this activity of production and consumption of data.

In completely different fields - physics and biology -, the scientist Ludwig von Bertalanffy developed in the 40s an open system theory [2]. Integrated into a General System Theory [3] an open system is characterized by a continual exchange of materials and energy, at the same time between the interior and the outside of the system, but also between its internal components. This dynamics allows a living organism to grow, develop, reproduce and to survive in spite of external changing conditions, while causing regeneration, renewal, destruction and replacement of the entities which compose it. The exchange of constituents which cannot be reduced to information provides the entity with a type of opening of a completely different nature. Applied to the software agent field, this opening of physical nature offers more important flexibility than the informational opening but leads to a structural modification of the

agent. This transformation must imperatively be controlled by equipping the agent with adapted internal mechanisms, under penalty of causing its stop.

The following section introduces a model of software agent which suits the study of the concept of physical opening.

2.2 Systemic Agent

Thanks to its broad applicability, the systemic approach [12], [4] provides - in the software agent context - the abstract tools allowing the consideration of the two types of openings considered previously: the informational opening and the physical opening. A *systemic agent* is a software agent (1) surrounded by an environment: that means that the agent is located in a universe where there are action and reciprocal reaction, (2) provided with a software architecture which changes in time, (3) in order to carry out a certain activity.

This last point - the activity of the agent - must be clarified. Indeed, when it is considered within its most general framework, the activity of the agent has a double aspect: the functioning related to the task which was assigned to it and the functioning bound to the preservation of its physical integrity. Then, the agent has a double finality: an *external finality* materialized in the form of a task to be carried out or of a service to be supplied and an *internal finality* which object is to maintain the internal integrity of the agent. The designer of systemic agent is then confronted with the following problem: how to manage conflicts that may occur between the two finalities. In other words, the decoupling of the two finalities poses a problem of arbitration which must be resolved at the level of the internal architecture of the agent. The selected solution is to provide the agent with a two-layered architecture, a level associated with the external finality and a level associated with the internal finality. In addition, when the expression of the two finalities leads to a conflict, it is the internal finality which is privileged.

The principle of a dual architecture posed, we must now specify its content that is to define the nature of the elements which make it up: the properties of the agent.

3 Attributes and Qualities

Many definitions and characterizations of the concept of agent were published in the specialized literature [8], [13]. Certain studies articulate their analyses around two principal axes: properties possessed by the agent and the application area considered [9]. Our previous works led us to keep only one of the two axes: properties. Two classes of properties are then distinguished: *attributes* and *qualities* [14].

3.1 Attributes

An attribute materializes a property of an agent which can be reduced to a mechanism, a perfectly known or customisable software device. For instance, the *mobility* attribute can be provided with a single parameter: the next site to be reached. Two pieces of information can customize the *replication* attribute: the rate of replication and the site of replication. The determination of the number and the meaning of the parameters of an attribute is made during the conception of the attribute independently from the

agent which will contain it. The decoupling between the realization of the attribute and the various considered parameter settings led us to associate a mechanism and one or several use policies to every attribute.

Two main attributes will be used within the open agent's framework: *mobility* and *transformation*. The mobility attribute is present in many mobile agent based applications [11], and will be used in section 5.2 to illustrate an aspect of the physical opening. The second attribute is specific to the model of agents which we develop: the Systemion Model. This model will be presented in the following section. Two additional attributes will be evoked but not explained in detail: *replication* and *perception*.

Mobility: mobility allows an agent to migrate to other sites. The notion of itinerary is often associated to mobility. An itinerary corresponds to an ordered list of hosts having to be visited by the agent, possibly specifying the actions to be carried out in the case of an unreachable or hostile host. An itinerary is characterized by a site of departure, a succession of intermediate sites and a site of arrival; in many mobile agent applications the site of arrival is the same as the site of departure; it is then called circular itinerary. Sometimes, the notion of route is not adapted because it is preferable in certain cases, to calculate at the last moment the next site to be visited (to reach for example the least loaded host).

Modification: the modification attribute can be interpreted in various manners. We will distinguish two variations from this attribute: evolution and metamorphosis. *Evolution* can be seen as a succession of gradual transformations, relatively slow and going in the same direction. This type of transformation is very studied in several fields of research such as adaptive systems and artificial life. In our work, we are interested more particularly in the abrupt transformation of a software agent. This type of transformation, that we call *metamorphosis*, is defined as a change of nature or structure of the agent, more or less radical. Contrary to evolution, metamorphosis is a fast and sudden change.

3.2 Qualities

Unlike the notion of attribute, a quality cannot be given a precise definition accepted by all the researchers. Autonomy and intelligence are two paradigmatic examples of what we understand by qualities. Qualities are difficult to measure as they are manifold. Consequently, there are various complementary models for a quality: for instance, there are several models of autonomy. Indeed, there is a great variety of points of view among the researchers who are interested in this property: autonomy can be considered either as a global property or as a partial property of the agent, as a social or not social characteristic of the agent [5], [7]. Qualities introduce complexity within the agent.

To illustrate one of the aspects of the physical opening of an agent, we will use a model of autonomy which considers it as a partial and not social property of the agent in section 5. This model derives from the definition: *autonomy is the condition of a person or a group that chooses its own laws*. In our context, policies or behaviours take the place of laws.

The next section synthesizes the various concepts presented previously, within a particular architecture of agent.

4 The Systemion Model

The *systemion model* (contraction of *systemic daemon*) is a particular model of systemic agent [cf section 2.2]. A *systemion* [14] is a software agent which integrates in its architecture, the concepts of internal finality and external finality, qualities, attributes and task. This architecture can be split up into two subsystems:

(1) a *functional subsystem* which materializes the external finality, i.e. properties (qualities and attributes) related to the fulfilment of the task possibly assigned to the agent. This task can change during the systemion life-cycle and constitutes the most flexible part of the systemion. Application complexity (including cooperation, coordination and communication interactions) is embedded into the task abstraction. To be able to integrate the first task in its functional subsystem or to change current function, a systemion must incorporate into its behavioural subsystem, the metamorphosis attribute

(2) a *behavioural subsystem*: it contains the properties (qualities and attributes) specific to the agent, that is the properties which are independent from the task which it will have to execute. This software layer implements the internal finality (Figure 1).

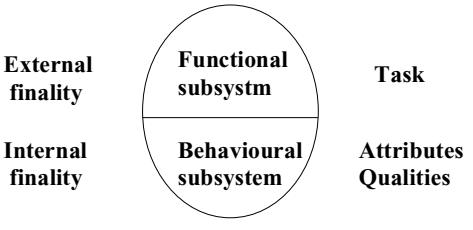


Fig. 1. Systemion Architecture

4.1 Tasks

The functional subsystem establishes the current competence of the systemion in the form of a task. It is convenient to consider a task as being constituted of one or several components, each one providing a distinct service or all the components fulfilling a single function. Using the modification attribute included in the behavioural subsystem, the functional subsystem of a systemion can be modified by executing predefined operations on one or several components which it contains. Four operations on a component are defined: (1) *adding*: the component is inserted into the functional part of the systemion and activated; we shall say that the component is active (2) *removing*: it is the reverse operation of the previous one; the component is removed by the functional subsystem (3) *freezing*: the component remains present in the functional part of systemion but is not executed until a new reactivation takes place; the code segment is frozen (4) *Activation*: the component is reactivated (Figure 2).

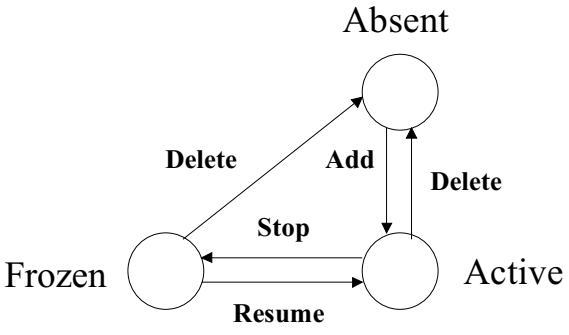


Fig. 2. Task modification

Being able to dynamically add a component to a systemion contributes to extend the rendered services (extensibility). Dynamically replacing a component by another allows to adapt it to the particular characteristics of a given host or to modify the service to be provided. Finally, temporary freezing of a component allows a mobile agent not to execute a component on one or several visited sites, without deeply modifying the systemion structure. Indeed, addition and removing of components can be expensive operations in CPU time and bandwidth.

4.2 Using the Systemion Model

Using the Systemion Model, we will analyze two classes of very different agents: *system daemons* and *worms*. This analysis will enable us to illustrate the concept of physical opening in the following section.

Daemons are present in all modern computer systems. They carry out background service functions such as the management of network connections and printing requests. For that, they continuously await the requests coming from the client processes, then carry out the requested service. From a systemion point of view, daemons possess only a functional subsystem because the properties they implement are only dedicated to the execution of the task which was assigned to them: daemons are *unitask agents*. Deprived of internal finality, the behavioural subsystem of a daemon is empty (Figure 3.a).

Without simplifying excessively, we shall say that the purpose of a worm is to replicate itself locally or remotely using a communication network. According to the systemion architecture a worm is constituted by a behavioural subsystem which integrates two attributes: a mechanism of perception (local perception and perception of remote hosts) and a mechanism of replication (local replication and remote replication), attributes which confer a certain mobility to it. Not executing any particular task, a worm doesn't integrate a functional subsystem, which is empty and fixed. Indeed, its functional subsystem will not undergo any operational modification during its lifespan (Figure 3.b).

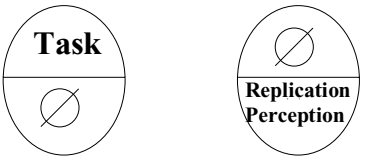


Fig. 3. (a) Daemon (b) Worm

5 Physical Opening

5.1 Open Functional Subsystem

The agents first studied were all functionally determined (empty or unitask). The systemion model supplies the tools necessary to the design and the implementation of agents functionally not determined. An agent of this type is provided with a functional subsystem which can vary in time, following the various tasks which it has to carry out. So that this dynamic modification of function can take place, the agent incorporates into its behavioural subsystem a suitable mechanism: the *modification* attribute presented in section 3.1. Not to jeopardize the systemion's existence the change of current task is triggered by the behavioural subsystem but the modifications are carried out in the functional subsystem. Thus it constitutes the most flexible part of the systemion.

Incorporating this internal transformation mechanism, the systemion can evolved in different ways. Let us illustrate this with an example. Figure 4 describes the following scenario: at time t a mobile agent arrives on host A. This systemion is an empty functionally not determined agent. It means that its functional subsystem does not contain any task to execute when host A receives it. At time t+1 the agent includes in its functional subsystem task T1 lying in a task repository. During its execution task T1 "asks" the systemion to migrate to site B. After the migration the systemion continues the execution of T1 on B, then deletes T1 from its functional subsystem and adds task T2 to it. These different operations are carried out using the ones presented in section 4.1. The systemion model enables to build *pluritask agents*.

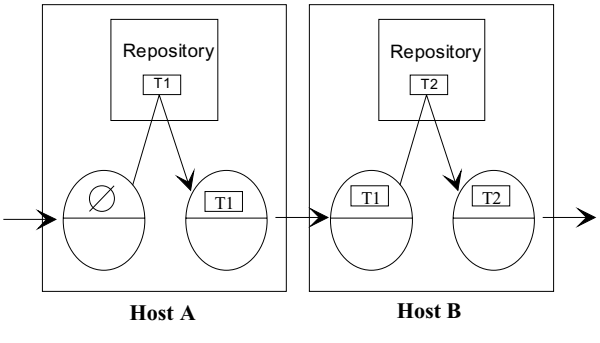


Fig. 4. Agent functionally not determined

A pluritask agent can be used in various situations: (1) an empty mobile systemion is used to discover a network or to test its environment, (2) a task code used at a given moment, may be no longer accurate in a while since the main characteristics of the local environment may change, so the agent deletes this unusable task code.

5.2 Open Behavioural Subsystem

The physical opening can be applied to the behavioural subsystem of the agent. As we have already explained about the functional subsystem, the aim of the physical opening is adding or removing behaviours having an external origin. The fundamental difference from task changing is that the modification(s) concern the part of the agent that ensures its reliability, i.e. the subsystem maintaining the stability of the agent. The physical opening of the behavioural subsystem introduces a risk for the agent's life.

We shall now illustrate how this opening can be done using the concepts presented above. The behavioural subsystem includes the attributes and qualities which define the essence and personality of the agent which are independent from the tasks that may be given to it. The concept of attribute differentiates the mechanism from the modalities of its use; each modality is called use policy or more simply policy. A particular model of autonomy is the autonomy with regard to an attribute. This model can be described as follows: : an agent is *autonomous with regard to an attribute* if it can rationally choose or randomly elect one policy among several ones associated to this attribute and can change current policy during its life.

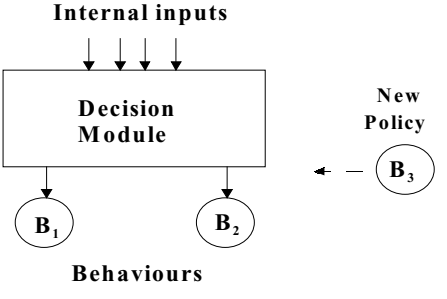


Fig. 5. An incoming new behaviour

In other words, an agent is autonomous with regard to an attribute if the policy it executes is not always explicitly forced on it by the environment. This kind of autonomy requires a set of behaviours and a decision module used by the agent to select the current policy (Figure 5). We do not intend to present the structure of the choice module in details. That is why it is represented by a black box. When a new external behaviour must be incorporated into the agent, it automatically results in modification within the choice module. It means that the choice module also has to be modified, even totally replaced (by using the physical opening).

If we take mobility as an example of attribute, different migration policies are possible: (1) random navigation, (2) navigation directed according to a special criterion

(e.g. a machine load). A mobile agent will include in its behavioural subsystem a choice module adapted to the handling of these two policies. If a new behaviour has to be taken into account by the agent, e.g. circular navigation, the choice module will also have to be modified. It can have various more or less serious consequences since policies have different complexities. Indeed, a behaviour can be made of one action: directed navigation and random navigation are mono-action policies. On the contrary, circular navigation is made of several elementary moves.

The physical opening of the behavioural subsystem is trickier than the physical opening of the functional subsystem. Anyway, the systemion model makes it possible to study both of them.

6 Conclusion

Complex systems and open systems have common characteristics that theories resulting from the systemic thought try to specify. By applying to the software agent the concepts of physical and informational opening and of two-layered architecture with stable/unstable contents, the Systemion Model has illustrated the interest of the dual reasoning in the study of complex computer systems.

References

1. Auyang, S. Y.: Foundations of Complex-System Theories in Economics, Evolutionary Biology, And Statistical Physics. Cambridge (UK), (1998).
2. Bertalanffy, L. von: The Theory of Open Systems in Physics and Biology. Science, 111 (1950), pp. 23-29.
3. Bertalanffy, L. von: General System Theory. George Braziller, New York, (1968).
4. Eriksson, D. M.: A Principal Exposition of Jean-Louis Le Moigne's Systemic Theory. Cybernetics and Human Knowing, Vol. 4, no 2-3, (1997).
5. 5. Franklin, S., Graesser, A.: Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages (ATAL 96)*, (Edited by Müller J. P., Wooldridge M. and Jennings N.), Lecture Notes in Artificial Intelligence, 1193, Springer Verlag, (1996).
6. Hewitt, C.: Offices Are Open Systems. ACM Transactions on Office Information Systems, Vol. 4, No.3, July 1986, pp. 271-287, (1986).
7. Hexmoor, H., Castelfranchi, C., Falcone, R. (eds.): Agent Autonomy. Kluwer Academic Publishers, Boston, (2003).
8. Jennings, N., Wooldridge, M.: Applications of Intelligent Agents. In *Agent Technology: Foundations, Applications, and Markets* (Edited by N. R. Jennings and M. Wooldridge) Springer Computer Science, (1998).
9. Jennings, N., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development. In *Autonomous Agents and Multi-Agent Systems*, 1, pp 275-306, Kluwer Academic Publishers, Boston, (1998).
10. Kephart, J. O., Chess, D. M.: The Vision of Autonomic Computing. IEEE Computer, Vol. 36, no 1, pp 41-50, (2003).
11. Milojicic, D.: Trend Wars – Mobile agent applications. *IEEE Concurrency*, Sept 1999, pp 80-90, (1999).

12. Le Moigne, J. L.: La théorie du système général – Théorie de la modélisation –. Ed. PUF, Paris, (1977).
13. Parunak, H. V. D., Breuckner, S., Sauter, J.: A Preliminary Taxonomy of Multi-Agent Interaction. In *Agent-Oriented Software Engineering (AOSE) IV*, P. Giorgini, Jörg Müller, James Odell, eds., Lecture Notes on Computer Science volume (forthcoming), Springer, Berlin, (2004).
14. Sanchis, E.: Designing new Agent Based Applications Architectures with the AGP Methodology. In *Proceedings of the twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (*WETICE* 2003), IEEE Computer Society, (2003).
15. Simon, H. A.: The Sciences of the Artificial. MIT Press, Cambridge (Massachusetts), (1996).
16. Wolfram, S.: Universality and Complexity in Cellular Automata. Physica D, 10, Jan 1984, pp 1-35, (1984).

Hybrid System Reachability-Based Analysis of Dynamical Agents

Eric Aaron

Department of Computer Science
Wesleyan University
Middletown, CT 06459
eaaron@wesleyan.edu

Abstract. This paper describes a hybrid dynamical system-based approach to formalizing and mechanizing analyses of dynamical agents, i.e., situated, embodied actors that continuously respond to their environment. As an example, the paper describes a class of formalized metrics for reasoning about the relative difficulties of agent navigation in various environments —not just whether one scenario is more difficult than another, but how much more difficult a scenario might be— and presents results of relative difficulty reasoning using a specific example metric. This illustrates that qualitative or heuristic agent properties, which are commonly unformalized and imprecise, may be formalized and rigorously analyzed using this approach. The paper also discusses the potential implementation of relative difficulty metrics in meta-intelligent agents.

1 Introduction

There are two interrelated, often challenging prerequisites for application-specific reasoning about intelligent agents: identifying a framework for performing the analysis; and, more fundamentally, representing and formalizing the ideas to be analyzed. Consider, for instance, a mission planning task with a robot that employs dynamic navigation (i.e., navigation that is determined by systems of differential equations), in which planners must send the robot from an unpredictable point in its *initial region*, the Western section of its environment, to a known target position in the Northeast corner of its world. As part of mission design, planners must select which of two world configurations the robot will experience. In one scenario, there are three obstacles, aligned North-to-South, halfway between the world's Western and Eastern boundaries; the other scenario contains those three obstacles plus a fourth, located in the Southeast quadrant (see Figure 1). Mission planners might find it useful to have mechanical assistance in computing the *relative difficulties* of the various scenarios—not just which one is more difficult, but *how much* more difficult one is than another. It may not be immediately apparent, however, how to formalize a notion of relative difficulty, much less how to analyze relative difficulty measures for the scenarios.

This paper describes a *hybrid dynamical system*-based approach to formalizing and mechanizing such analyses of *dynamical agents*, i.e., situated, embodied

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 233–244, 2006.

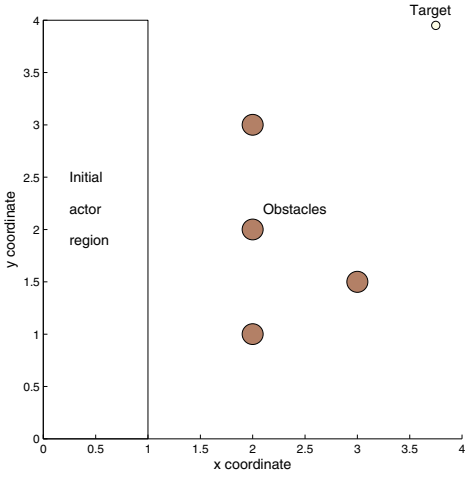


Fig. 1. One world used for experiments. It contains four stationary obstacles, a target, and an *initial region*, $x, y \in [0\,..\,1, 0\,..\,4]$. Does the obstacle at (3,1.5) add to navigation difficulty?

actors, either material (mobile robots) or virtual (animated characters), that continuously, intelligently respond to their environment. Numerous architectures for autonomous agents and intelligent systems are described as "hybrid," a designation often referring to their combination of discrete (symbolic) and continuous (analog) dynamics. The theory of hybrid dynamical systems (*hybrid systems*, for short) applies to precisely these kinds of systems; past domains of application for hybrid system models include descriptions of air-traffic management systems [1], manufacturing systems [2], and robots [3] and animated actors [4,5]. Typically, properties of hybrid systems are specified as modal logic formulae; of particular importance are *safety properties*, which specify that the system never reaches a "bad" state (e.g., a collision state in which an actor's position equals an obstacle's position). Verification of such properties is commonly performed by *reachability analysis* [6]: System executions are constrained to begin from some point in a pre-specified initial region, and a search procedure computes all states reachable from that initial region by that system. If no "bad" state is reachable, the desired safety property necessarily holds throughout any run of the system. There are severe restrictions on decidability of reachability of hybrid systems [7]. Correspondingly, there are several methods for approximating hybrid systems and thus making analysis feasible, including methods based on *flow pipes* [8,9,10] or ellipsoidal approximations [11].

The utility of reachability-based analysis can extend even beyond properties with strict, conventionally accepted specifications (e.g., collision avoidance). By introducing elements of heuristic judgment into reachability analysis, some qualitative or heuristic properties of intelligent agents may be formalized and standardized in a framework that supports efficient mechanical assistance in

verification. As a concrete example, this paper presents a rigorous, reachability-based approach to quantifying relative difficulties of some navigation scenarios.

Given the geography of a model —including the initial region and the locations of the actors, the obstacles, and the target— and a dynamical system (i.e., differential equations) describing how that model will evolve over time, the relative difficulty analysis procedure does the following:

1. Creates an abstraction of that dynamical model by decomposing the domain space of the dynamical system into *simplices*;
2. Computes a piecewise linear approximation of the model, i.e., for each simplex, constructs a linear approximation of the dynamics over that simplex;
3. Treating each simplex as a state, performs *state reachability analysis* on the approximated, abstracted model, estimating the likelihood that the agent will reach its target before colliding with an obstacle;
4. Utilizes the results of reachability analysis to quantify the relative difficulty of that modeled world.

This paper further describes the procedure for relative difficulty analysis of dynamic navigation in fixed environments. In addition, the paper presents demonstrations involving a specific example metric, and it discusses the potential for relative difficulty metrics in *metacognitive* agents capable of reasoning about their own navigation intelligence.

2 Methodology

2.1 A Reactive Model for Navigating Actors

This section introduces the dynamical navigation system employed in the illustrative examples of section 3.2, a scalable, adaptive approach to modeling multiple autonomous agents in dynamic environments of multiple obstacles and targets [12]. At the core of the mathematics underlying navigation are non-linear *attractor* and *repeller* functions that represent the targets and obstacles (respectively). Another non-linear system combines their weighted contributions in calculating an actor's angular velocity, dynamically adapting to real-time changes in the environment. Together, these non-linear systems form the navigation functions for the actors, generating natural-appearing motion while avoiding collisions and other undesirable behaviors. The agent heading angle ϕ is computed from a non-linear dynamical system of the form

$$\dot{\phi} = f(\phi, \mathbf{env}) = |w_{tar}| f_{tar} + |w_{obs}| f_{obs} + n, \tag{1}$$

where f_{tar} and f_{obs} are the attractor and repeller functions for the system, and w_{tar} and w_{obs} are their respective weights on the agent. (n is a noise term, which helps avoid local minima in the system.)

The weights themselves are determined by computing the fixed points of the non-linear system

$$\begin{cases} \dot{w}_{tar} = \alpha_1 w_{tar}(1 - w_{tar}^2) - \gamma_{12} w_{tar} w_{obs}^2 + n \\ \dot{w}_{obs} = \alpha_2 w_{obs}(1 - w_{obs}^2) - \gamma_{21} w_{obs} w_{tar}^2 + n \end{cases}, \tag{2}$$

where the α and γ parameters are designed to reflect stability conditions of the system. Other parameters are also concealed in the terms presented above. For instance, a repeller function f_{obs} depends on parameters that determine how much influence obstacles will have on an actor.

This is only an overview of one significant part of the navigation system; there is considerably more detail, including applications to three-dimensional environments [12]. This overview gives a feel for the mathematics, however, suggesting the complexity of reasoning about it.

2.2 Simplicial Decomposition

This method of relative difficulty reasoning relies on linear approximations to navigation functions. For the examples in this paper, that entails approximating the functions for $\dot{\phi}$ given in section 2.1, as well as functions for $\dot{x}$ and $\dot{y}$ that derive from ϕ. For other navigation methods, similar ideas would apply.

To arrive at linear approximations of such functions over a rectangular, bounded, n-dimensional domain space, that domain is divided into *n-simplices*, where an *n-simplex* (or just *simplex*, when n is understood) is the region of n-space defined by the convex hull of $n+1$ affinely independent points. For efficiency of detecting adjacent simplices, the decomposition should be such that each simplex is adjacent to at most one other simplex across each *facet* (i.e., face of dimension $n-1$).

The $n+1$ vertices of each simplex are used to generate a linear approximation to an original navigation function. That is, the navigation function over $\mathbb{R}^n$ is evaluated at the $n+1$ vertices, and each evaluation leads to a linear equation in $n+1$ variables. Together, the $n+1$ evaluations yield a system of $n+1$ equations in $n+1$ variables, and the unique solution of this system provides the coefficients for the linear approximation over that simplex. This can be straightforwardly accomplished, although the optimal structure for this application remains an open question.

2.3 Reachability Computation of Approximated Dynamics

In broad terms, the approach to reasoning about navigation is based on *reachability analysis* [6]. Reachability analysis is commonly employed to verify properties of some classes of dynamical or hybrid dynamical systems. In such a verification, the system model is decomposed into discrete states, and each state is identified as either satisfying or falsifying particular properties of the system (e.g., whether the values of variables are within a certain prescribed range). To verify that a model satisfies a given property, an automated tool such as [8,13] computes (often with conservative over-estimation) whether the system could evolve into a state that falsifies the property. If no such falsifying state is reachable, then the system must always satisfy the property.

A similar state reachability approach is employed in the method presented here; undecidability restrictions that apply to reasoning about complex dynamics are avoided by instead reasoning about approximations to the original navigation system, i.e., the piecewise linear system induced by the simplicial decomposition

of section 2.2. For purposes of reachability, each simplex is considered to be a state. Then, a state is reached if the system evolves into that state, as controlled by the navigation functions.

As an example of reachability over n-space, let A stand for the $(n+1)\times(n+1)$ coefficient matrix of the linear dynamical system in a state S_i. To determine if the system could evolve from S_i to an adjacent state S_j across the unique shared facet $F_{i,j}$, A is evaluated at each vertex of $F_{i,j}$, and an n-vector of the dynamics at that vertex is computed, with each element of the vector being the time-derivative of the corresponding system variable. For example, in a three-dimensional case, where the system is defined by the evolution of the values of ϕ, x, and y, a computed 3-vector would be $\langle\dot{\phi},\dot{x},\dot{y}\rangle$.

Using these computations of the induced vector field of A, it is determined at each vertex of a facet if the field is directed outward or inward with respect to the simplex by taking the dot product of the dynamics with an outward normal vector of the facet being considered. That is, for a facet $F_{i,j}$ with vertices $\{x_1,\ldots,x_n\}$, let $onv_{F_{i,j}}$ be an outward normal vector for $F_{i,j}$; note that it is an outward normal at every x_k. To simplify notation, also let $V_{i,j,k}$ stand for the n-vector of system dynamics at vertex x_k of facet $F_{i,j}$. To compute whether the dynamics are directed outward at a vertex x_k, simply consider the sign of the dot product $V_{i,j,k}\cdot onv_{F_{i,j}}$. If that quantity is positive, the dynamics are directed outward; similarly, if that quantity is negative, the dynamics are directed inward at that vertex.

If the dynamics are directed inward at *every* vertex of the facet, then because the dynamics are *linear*, there can be no point on the facet at which the dynamics are directed outward, and thus there is no point at which the system could evolve outward through that facet. For facets on which a crossing is possible —i.e., the dynamics are directed outward for at least one vertex— the relative difficulty analysis assigns a likelihood that the system would evolve across that facet, based on the number of vertices at which dynamics are directed outward and the magnitude of outward-directed force. These two factors are the bases for selecting which state transitions to consider in the reachability analysis: Likely transitions are included in the computation while unlikely transitions are pruned from consideration, enabling efficient and effective relative difficulty measures.

3 An Approach to Quantifying Relative Difficulty

For the purposes of relative difficulty analysis, below, a state is considered to contain an actor (or obstacle, or target) when some relevant, user-specified portion of its x-y projection is contained within the x-y projection of the state; for the examples in section 3.2, for instance, containment means that the center point of an entity is within a state. Then, if the system is in a state containing both an actor and an obstacle, a collision is considered to have occurred between that actor and that obstacle. Similarly, if the system is in a state containing an actor and its target, that actor has reached its target.

Throughout this section, fixed environments of non-moving obstacles and targets are assumed, although as mentioned briefly in section 4, the underlying theory also applies to dynamic worlds.

3.1 A Framework for Metrics of Relative Difficulty

Given an agent and its dynamical navigation system (e.g., the one in section 2.1), and given an *initial region* of the world that bounds the system's initial state (e.g., $\phi, x, y \in [-\pi \,..\, \pi, 0 \,..\, 1, 0 \,..\, 4]$), relative difficulty analysis is based on approximated likelihoods that an agent will collide with an obstacle before reaching its target. Intuitively, the difficulty of that world is directly related to those likelihoods.

As essential components, relative difficulty analysis employs the state reachability method of section 2, and notions of *bad* and *good* states. A bad state represents a localized judgment that, for an actor in that state, collision is more likely than reaching a target: A state S is bad if either state S contains an obstacle or, when the system is in state S, it is more likely to evolve into a bad state than a good state as its next (i.e., immediate successor) state. Analogously, a good state represents a localized judgment that reaching a target is more likely than collision, and it has a similar, inductive-feeling conceptualization. A *neutral* state is one from which neither collision nor target achievement is more likely, including potential cases where an actor would never collide nor reach its target.

A state reachability analysis assigns values to states, which are then used to compute relative difficulty. The reachability analysis begins with all *initial states*, states that are at least partially contained in the initial region. Proceeding from them to all states that might be reachable by the system, states in the initial region are assigned a value by a stack-based, depth-first traversal of the overall state space. At the end of the procedure, all states in the initial region will be assigned either *good*, *bad*, or *neutral* values.

Once all states in the initial region have been assigned values, relative difficulty is computed as the difference:

$$reldiff = safeweight * \frac{sb}{sz} - liveweight * \frac{sg}{sz}. \tag{3}$$

The variables refer to the following quantities / parameters:

- sz is the total number of states in the initial region.
- sb is the number of states assigned value *bad*.
- sg is the number of states assigned value *good*.
- *safeweight* is a user input parameter indicating how much emphasis to place on the system not entering a bad state. (It is called *safeweight* in reference to general safety properties of systems.)
- *liveweight* is a user input parameter indicating how much emphasis to place on the system eventually entering a good state. (It is called *liveweight* in reference to liveness properties of systems.)

If both *weights* are positive, a greater number indicates more *bad* states and greater difficulty, whereas a lower number indicates more *good* states and lower

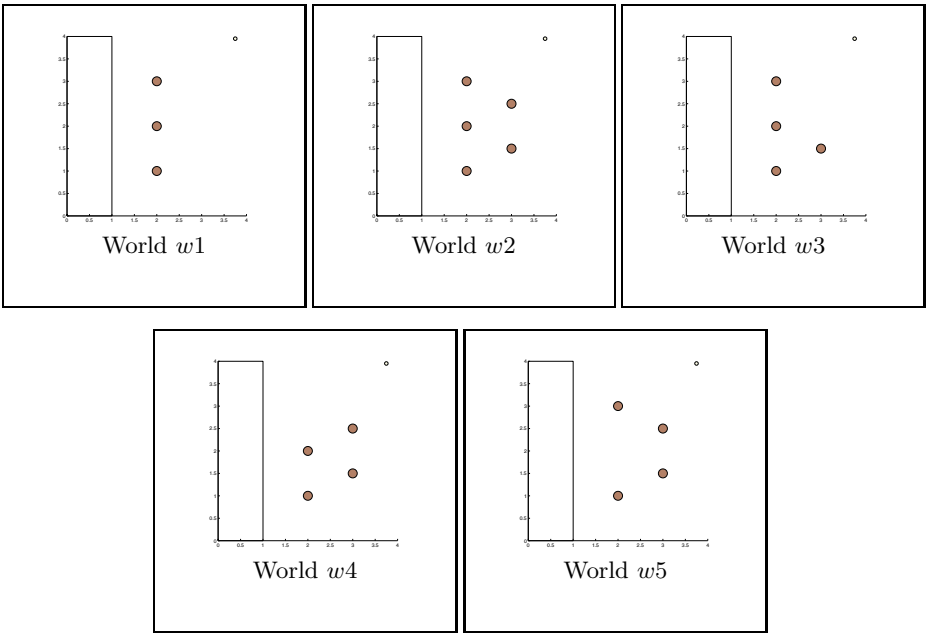


Fig. 2. The five worlds used for experiments. Obstacle configurations vary from world to world. The example initial region and target location in each world, as well as the axes, are identical to those in the example world detailed in Figure 1.

difficulty. It is not an absolute difficulty measure, but it can enable more educated judgments about navigation missions.

3.2 Applications and Demonstrations

This section presents and discusses experimental analyses of navigation missions in simulated worlds. For these examples, the weights *safeweight* and *liveweight* were 1, the actors' size was 0.05, the actors' forward velocity was 0.2, and the domain space was divided into 2^{13} sub-boxes.

The image in Figure 1 represents an example world configuration used to illustrate the methodology for computing relative difficulty. All entities are drawn as circles. The decision that x, y coordinate spaces would be $x, y \in [0\,..\,4, 0\,..\,4]$ made expressing obstacle locations convenient but was otherwise arbitrary, simply creating some bounded region to decompose into states.

For the actors' dynamic navigation in these worlds, the method presented in section 2.1 was intentionally tuned so that some actors would collide with obstacles. It was possible to tune navigation so that collisions never occurred, but such optimal performance would not yield interesting data from the perspective of relative difficulty.

Experiments were performed on five different virtual world configurations (see Figure 2), which differed in the positions of stationary obstacles. The target

Initial (Sub-)Regions	*Worlds*				
	$w1$	$w2$	$w3$	$w4$	$w5$
$\phi \in [-\pi \,..\, \pi], x \in [0 \,..\, 1], y \in [0 \,..\, 4]$	-0.477	-0.387	-0.477	-0.649	-0.623
$\phi \in [-\pi \,..\, \pi], x \in [0 \,..\, 1], y \in [0 \,..\, 2]$	-0.263	-0.098	-0.263	-0.357	-0.534
$\phi \in [-\pi \,..\, \pi], x \in [0 \,..\, 1], y \in [2 \,..\, 4]$	-0.724	-0.724	-0.724	-0.995	-0.724
$\phi \in [-\pi \,..\, \pi], x \in [0 \,..\, 1], y \in [1 \,..\, 3]$	-0.426	-0.426	-0.426	-0.874	-0.548
$\phi \in [-\pi \,..\, \pi], x \in [0 \,..\, 1], y \in [3 \,..\, 4]$	-0.935	-0.935	-0.935	-0.994	-0.935
$\phi \in [\frac{-\pi}{2} \,..\, \frac{\pi}{2}], x \in [0 \,..\, 0.5], y \in [3 \,..\, 4]$	-0.984	-0.984	-0.984	-1	-0.984

Fig. 3. Relative difficulty results for six different initial regions in each of the five different world configurations in Figure 2. In principle, values could range from -1 (least difficult) to 1 (most difficult); negative values indicate greater likelihood of reaching the target than colliding with an obstacle.

location was the same in all experiments. The outermost possible bounds on actors' initial regions were the same in all worlds, as well, but many experiments examined sub-regions as initial regions, to explore the role of starting position on navigation difficulty.

Using the metric given by (3), thirty relative difficulties were computed (see Figure 3). Six different sub-regions of $[0 \,..\, 1, 0 \,..\, 4]$ were chosen as possible initial regions for the actors, and relative difficulty values were computed for each possible initial region in each of the five world configurations.

A General Result. One common consideration is the role of the initial region in relative navigation difficulty. Would difficulty vary if actors were restricted to start only in some sub-portion of the initial region $x, y \in [0 \,..\, 1, 0 \,..\, 4]$? In particular, it seems intuitive that actors starting in sub-regions with greater y values (e.g., $x, y \in [0 \,..\, 1, 2 \,..\, 4]$ or $x, y \in [0 \,..\, 1, 3 \,..\, 4]$) would have a clearer path to the target and hence an easier navigation task.

Experimental results are consistent with this intuition. When considering the full range $y \in [0 \,..\, 4]$ or the subrange $y \in [0 \,..\, 2]$, the relative difficulty results are substantially greater than over $y \in [2 \,..\, 4]$ or $y \in [3 \,..\, 4]$. Furthermore, when considering only $y \in [3 \,..\, 4]$, every world except world $w4$ had relative difficulty value -0.935, substantially easier still. World $w4$ lacked the obstacle at position $(2, 3)$ common to all other worlds, and its relative difficulty readings indicated even *easier* navigation: -0.995 over $y \in [2 \,..\, 4]$, and -0.994 over $y \in [3 \,..\, 4]$, with 0 bad states indicated in both conditions. Relative difficulties for world $w4$ were not a "perfectly un-difficult" -1 because some states were marked as neutral, presumably due to actors going out-of-bounds. This also explains why the value for $y \in [2 \,..\, 4]$ was very slightly lower (indicating less relative difficulty) than for $y \in [3 \,..\, 4]$: A lower *percentage* of the states may have been judged likely to drive actors out of bound in the larger, less-cornered region.

Simulation data are also consistent with these results. Simulations of 1000 actors on worlds $w1$ and $w4$ suggest that $y \in [0 \,..\, 2]$ was more difficult than $y \in [0 \,..\, 4]$ but less difficult than $y \in [2 \,..\, 4]$, which was in turn less difficult

than $y \in [3\,..\,4]$. Simulations on $w4$ also support the observations made above regarding collision-free initial regions and the greater out-of-bounds likelihood for the more-cornered initial region. Evaluation of results is discussed further in section 3.3.

Experiment 1: Removing Back-Row Obstacles. When considering all possible actors starting in initial region $x, y \in [0\,..\,1, 0\,..\,4]$ of worlds $w1$, $w2$, and $w3$, it is immediately apparent that the three obstacles at $x = 2$ are a barrier to collision-free target achievement. Given that permeable barrier, it is less immediately apparent how much the obstacles at $x = 3$ impede target achievement. To test this, the three worlds were compared, thus comparing cases with 0, 2, and 1 back-row obstacles, respectively, as indicated by the pictures in Figure 2. Any observed differences are directly due the absence of back-row obstacles.

On all initial regions tested, worlds $w1$ and $w3$ were *identical*. This is a striking, unexpected but not counter-intuitive result, answering a question from the introduction of this paper: The obstacle at coordinates $(3, 1.5)$ made no difference at all to navigation difficulty. This result is consistent with simulation data.

Results also indicate that the impact of the obstacle at $(3, 2.5)$ was present but not pervasive. On all initial regions contained within $y \in [1\,..\,4]$, the relative difficulty measures were identical across $w1$, $w2$, and $w3$: Neither back row obstacle affected actors in those states. On initial regions containing $y \in [0\,..\,1]$, however, world $w2$ was substantially more difficult. These results about the obstacle at $(3, 2.5)$ are intuitive, and they are consistent with simulation data of navigation with approximated dynamics.

3.3 Experiments: Discussion and Evaluation

In interpreting experimental results over various initial regions, recall that actors are generally considered over the *full range* of possible heading angles. In particular, this implies no restriction to only actors that are initially pointed in the direction of their target. Simulation data show a substantial number of actors that wind up out-of-bounds.

Because there is no accepted, objective criterion for relative difficulty, evaluation of the experimental results of section 3.2 is inevitably inexact. In this paper, results are evaluated, however loosely, based on two criteria: informal intuition about the meaning of relative difficulty in context, and comparison with collision data in simulated trials. In particular, collision data were the results of simulating 1000 actors randomly placed in the initial regions; the numbers of actors that collided or reached targets varied by as much as 5-10% from trial to trial. An experimental result is thus noted to be consistent with simulation data if experimentally determined likelihoods of collision and target acquisition are within 5-10% of simulation results.

4 Future Directions and Discussion

The framework for formalizing and quantifying relative navigation difficulty is intended to be broadly applicable and extendable. Details can be altered to

arrive at different relative difficulty metrics, tuned to particular applications. In addition, this framework is generally independent of the particular dynamical systems governing navigation. No matter how complex the underlying differential equations, the simplicial decomposition generates a continuous, piecewise linear hybrid dynamical system to which reachability analysis can be applied.

With this flexibility, the framework applies not just to different obstacle / target scenarios but also to different navigation methods. Up to the precision of the linear approximation performed to support reachability analysis, the framework could quantify relative difficulty differences among navigation methods in the same geography. It even allows for testing of the relative effects of altering parameter settings within a particular navigation method.

Methodology refinements and extensions are also possible on several levels. For instance, the details of the state-reachability search at the heart of the metric could employ a more complex constraint than the straightforward *bad-likelihood* > *good-likelihood*. In addition, the present method for computing relative difficulty theoretically extends to spaces of high dimensionality, which means it can be naturally extended to apply to *dynamic worlds* in which there are moving obstacles or targets. Even in its current form, however, the framework directly applies to some cases with incompletely specified or non-fixed environments, such as missions in which an actor pursues multiple targets in sequence, or where the goal is a *target region* rather than at a specific point.

4.1 Implementing Meta-intelligence over Dynamic Intelligence

Two levels of analysis have been described in previous sections. On one level, actors perceive the world around them and intelligently, dynamically adjust their navigation behavior (e.g., "turn more sharply now"). On another level, relative difficulty analysis reasons about the way that actors intelligently respond to their worlds, deriving judgments of how difficult one navigation scenario is when compared to others (e.g., "using the current metric, scenario A is 0.2 more difficult than scenario B"). As part of its procedure, the relative difficulty analysis employs approximations of the sub-propositional, reflex-level intelligence of the actors as bases for propositional, meta-level judgments.

It seems conceptually straightforward to implement this framework of meta-level reasoning in a dynamically intelligent agent, thus creating a meta-intelligent agent. The concrete examples in this paper were in the context of proposition-level reasoning being performed by an outside intelligence, but the relevant meta-level analysis does not require any information, meta-logical complexity, or computational power beyond that which an animated actor or mobile robot could already reasonably possess. For such meta-intelligence to be feasible, the agent would need access to only the relative difficulty functions (and data structures) and their arguments. Access to the functions themselves is straightforward to provide. The functions' arguments are simply the velocity of the agent itself and the positions and sizes of all entities that affect the agent's perception and navigation. For animated characters, such information is immediate; for mobile robots, mapping and localization concerns are significant but no more burden-

some in this context of meta-intelligence than in other applications. Therefore, in dynamical agents, a first-level meta-intelligence that arises from reasoning about sub-propositional intelligence (as exemplified by the relative difficulty reasoning) is implementable without inherent conceptual or semantic difficulties. Interestingly, this kind of meta-intelligent agent could reason about its reactive navigation process using *exactly the same analysis methods* that human observers do. Agents could use such meta-level information as part of improved heuristic wayfinding, incorporating low-level navigation intelligence with relative difficulty analysis, map management, and other higher-level path planning.

Furthermore, note that there is nothing exclusive to the example dynamical intelligence or relative difficulty reasoning presented in this paper that is necessary for such meta-intelligent unification. As long as the real-time intelligence is expressed by differential equations and the meta-level intelligence is directly expressed as reachability analysis, the implementation of meta-intelligence over dynamical intelligence seems to remain conceptually and semantically straightforward. Indeed, there is no restriction of the object domain even to be about navigation intelligence; the fundamental topics could be any dynamically changing aspects of embodied agents, such as intensities of biases, emotions, affinities or disinclinations toward elements of the world—anything that could be modeled over time by dynamical systems, given environmental inputs. Supplying a reachability-based semantics for meta-level knowledge does not always seem straightforward, but in some cases, it could be done for both state-oriented properties of variable values in a world (e.g., "Is it ever true that an agent prefers cold to heat?") and for more global properties of worlds overall (e.g., "Is the development of cold-aversion more likely in world $w1$ or world $w2$?").

5 Conclusion

It is not the goal of this paper to provide a single definition of relative difficulty for all situations. Instead, the paper suggests that the formal yet intuitive approach presented here could offer adaptable assistance in creating and deploying intelligent dynamical agents, useful in a variety of circumstances. Example applications of relative difficulty analysis suggest that even simple metrics can correspond well with intuition and provide formalized insights into differences among mission scenarios. In addition, the paper discusses the possibility of implementing this approach to achieve meta-intelligent relative difficulty analysis over intelligent wayfinding; the underlying theory can be generalized to other instances of both the dynamical intelligence and state reachability-based meta-intelligence levels.

Acknowledgements

Thanks to: George Hulley, Franjo Ivančić, Zachary Dodds, Michael Dickerson, Anil Nerode, Dimitris Metaxas, Viorel Mihalef, and Norm Badler.

References

1. Tomlin, C., Pappas, G.J., Sastry, S.: Conflict resolution for air traffic management: A study in muti-agent hybrid systems. IEEE Transactions on Automatic Control **43**(4) (1998) 509–521
2. Pepyne, D., Cassandras, C.: Hybrid systems in manufacturing. Proceedings of the IEEE **88** (2000) 1108–1123
3. Egerstedt, M., Hu, X.: A hybrid control approach to action coordination for mobile robots. Automatica **38**(1) (2002) 125–130
4. Aaron, E., Ivančić, F., Metaxas, D.: Hybrid system models of navigation strategies for games and animations. In: Proceedings of Hybrid Systems : Computation and Control. Volume 2289 of Lecture Notes in Computer Science. Springer Verlag (2002) 7–20
5. Aaron, E., Sun, H., Ivančić, F., Metaxas, D.: A hybrid dynamical systems approach to intelligent low-level navigation. In: Proceedings of Computer Animation. (2002) 154–163
6. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (2000)
7. Alur, R., Henzinger, T., Lafferriere, G., Pappas, G.: Discrete abstractions of hybrid systems. Proceedings of the IEEE **88** (2000) 971–984
8. Asarin, E., Dang, T., Maler, O.: The d/dt tool for verification of hybrid systems. In: Proceedings of Conference on Computer Aided Verification (CAV'02). Volume 2404 of Lecture Notes in Computer Science., Springer Verlag (2002) 365–370
9. Chutinan, A., Krogh, B.: Verification of polyhedral invariant hybrid automata using polygonal flow pipe approximations. In: Proceedings of Hybrid Systems : Computation and Control. Volume 1569 of LNCS. Springer Verlag (1999) 76–90
10. Dang, T.: Verification and Synthesis of Hybrid Systems. PhD thesis, Verimag, Institut National Polytechnique de Grenoble (2000)
11. Kurzhanski, A., Varaiya, P.: Ellipsoidal techniques for reachability analysis. In: Proceedings of Hybrid Systems : Computation and Control. Volume 1790 of Lecture Notes in Computer Science. Springer Verlag (2000) 202–214
12. Goldenstein, S., Karavelas, M., Metaxas, D., Guibas, L., Aaron, E., Goswami, A.: Scalable nonlinear dynamical systems for agent steering and crowd simulation. Computers And Graphics **25**(6) (2001) 983–998
13. Alur, R., Grosu, R., Lee, I., Sokolsky, O.: Compositional refinement for hierarchical hybrid systems. In: Hybrid Systems : Computation and Control. Volume 2034 of Lecture Notes in Computer Science. Springer Verlag (2001) 33–48

Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios

Suranga Hettiarachchi, William M. Spears, Derek Green, and Wesley Kerr

University of Wyoming, Laramie WY 82071, USA

Abstract. This paper introduces a novel framework for designing multi-agent systems, called "Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios" (DAEDALUS). Traditional approaches to designing multi-agent systems are offline (in simulation), and assume the presence of a global observer. In the online (real world), there may be no global observer, performance feedback may be delayed or perturbed by noise, agents may only interact with their local neighbors, and only a subset of agents may experience any form of performance feedback. Under these circumstances, it is much more difficult to design multi-agent systems. DAEDALUS is designed to address these issues, by mimicking more closely the actual dynamics of populations of agents moving and interacting in a task environment. We use two case studies to illustrate the feasibility of this approach.

1 Introduction

Engineering multi-agent systems is difficult due to numerous constraints, such as noise, limited range of interaction with other agents, delayed feedback, and the distributed autonomy of the agents. One potential solution is to automate the design of multi-agent systems in simulation, using evolutionary algorithms (EAs) [2,9]. In this paradigm, the EA evolves the behaviors of the agents (and their local interactions), such that the global task behavior emerges. A global observer monitors the collective, and provides a measure of performance to the individual agents. Agent behaviors that lead to desirable global behavior are hence rewarded, and the collective system is gradually evolved to provide optimal global performance.

There are several difficulties with this approach. First, a global observer may not exist. Second, some (but not all) agents may experience some form of reward for achieving task behavior, while others do not. Third, this reward may be delayed, or may be noisy. Fourth, the above paradigm works well in simulation (offline), but is not feasible for real-world online applications where unexpected events occur. Finally, the above paradigm may have difficulty evolving different individual behaviors for different agents (heterogeneity vs homogeneity).

In this paper we propose a novel framework, called "Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios" (DAEDALUS), for engineering multi-agent systems that can be used either offline or online. We will explore how DAEDALUS can be used to achieve global aggregate behavior, by examining two case studies.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 245–256, 2006.

2 Distributed Agent Evolution with Dynamic Adaptation to Local Unexpected Scenarios

With the DAEDALUS paradigm, we assume that agents (whether software or hardware) move throughout some environment. As they move, they interact with other agents. These agents may be of the same species or of some other species [6]. Agents of different species have different roles in the environment. The goal is to evolve agent behaviors and interactions between agents, in a distributed fashion, such that the desired global behavior occurs.[1]

Let us further assume that each agent has some procedure to control its own actions, in response to environmental conditions and interactions with other agents. The precise implementation of these procedures is not relevant, thus they may be programs, rule sets, finite state machines, real-valued vectors, force laws, or any other procedural representation. Agents have a sense of self-worth, or "fitness". Agents that experience direct performance rewards have higher fitness. Other agents may not experience any direct reward, but may in fact have contributed to the agents that did receive direct reward. This "credit assignment" problem can be addressed in numerous ways, including the "bucket brigade" algorithm or the "profit sharing" algorithm [3]. Assuming that a set A of agents has received some direct reward, both algorithms provide reward to the set B of agents that have interacted (and helped) those in A. Further trickle-back rewards are also given to those agents in set C that helped those in B, and so on. Agents that receive no rewards lose fitness. If fitness is low enough, agents stop moving or die.

Evolution occurs when individuals of the same species interact. Those agents with high fitness give their procedures to agents with lower fitness. Evolutionary recombination and mutation provide necessary perturbations to these procedures, providing increasing performance and the ability to respond to environmental changes. Different species may evolve different procedures, reflecting the different niches they fill in the environment.

3 Transition of Offline Applications to DAEDALUS

Our prior applications of EAs to design multi-agent systems have used the offline approach – a global observer assigns fitness to agents based on their collective behavior. In the next section, we show how DAEDALUS could be applied to two different applications, namely, obstacle avoidance and the self assembly of machines, in an online environment. For both applications, recombination and mutation operators provide the ability to respond to environmental changes (which can include the addition and/or removal of agents).

3.1 Obstacle Avoidance

In prior work we have shown how our artificial physics framework can be used to self-organize swarms of mobile robots into hexagonal lattices (networks) that

[1] The work by [8] is conceptually similar and was developed independently.

Fig. 1. Seven robots form a hexagon, and move towards a light source

move towards a goal (see Figure 1). We extended the framework to include motion towards a goal through an obstacle field. An offline EA evolved an agent-level force law, such that robots maintained network cohesion, avoided the obstacles, and reached the goal. The emergent behavior was that the collective moved as a viscous fluid [4].

The Artificial Physics Framework: In our artificial physics (AP) framework [7], virtual physics forces drive a swarm robotics system to a desired configuration or state. The desired configuration is one that minimizes overall system potential energy, and the system acts as a molecular dynamics ($\boldsymbol{F} = m\boldsymbol{a}$) simulation.

Each robot has position $\boldsymbol{p}$ and velocity $\boldsymbol{v}$. We use a discrete-time approximation of the continuous behavior of the robots, with time step Δt. At each time step, the position of each robot undergoes a perturbation $\Delta \boldsymbol{p}$. The perturbation depends on the current velocity, i.e., $\Delta \boldsymbol{p} = \boldsymbol{v}\Delta t$. The velocity of each robot at each time step also changes by $\Delta \boldsymbol{v}$. The change in velocity is controlled by the force on the robot, i.e., $\Delta \boldsymbol{v} = \boldsymbol{F}\Delta t/m$, where m is the mass of that robot and $\boldsymbol{F}$ is the force on that robot. F and v denote the magnitude of vectors $\boldsymbol{F}$ and $\boldsymbol{v}$. A frictional force is included, for self-stabilization.

From the start, we wished to have our framework map easily to physical hardware, and our model reflects this design philosophy. Having a mass m associated with each robot allows our simulated robots to have momentum. Robots need not have the same mass. The frictional force allows us to model actual friction, whether it is unavoidable or deliberate, in the real robotic system. With full friction, the robots come to a complete stop between sensor readings and with no friction the robots continue to move as they sense. The time step Δt reflects the amount of time the robots need to perform their sensor readings. If Δt is small, the robots get readings very often, whereas if the time step is large, readings are obtained infrequently. We have also included a parameter F_{max}, which provides a necessary restriction on the acceleration a robot can achieve. Also, a parameter V_{max} restricts the maximum velocity of the robots (and can always be scaled appropriately with Δt to ensure smooth path trajectories).

In this paper we are investigating the utility of a generalized Lennard-Jones (LJ) force law (which models forces between molecules and atoms).

$$F = 24\epsilon \left[\frac{2dR^{12}}{r^{13}} - \frac{cR^5}{r^7} \right] \tag{1}$$

$F \leq F_{max}$ is the magnitude of the force between two robots i and j, and r is the distance between the two robots. R is the desired separation between robot i and all other neighboring robots. The variable ϵ affects the strength of the force, while c and d control the relative balance between the attractive and repulsive components. In order to achieve optimal behavior, the values of ϵ, c, d, and F_{max} must be determined. Our motivation for using the LJ force law is that (depending on the parameter settings) it can easily model crystalline solid formations, liquids, and even gases.

Optimizing Parameters Using Genetic Algorithms (Offline Approach): Given the generalized force law, such as LJ, it is necessary to optimize the parameters to achieve the best performance. We achieve this task using a genetic algorithm (GA). Genetic algorithms are optimization algorithms inspired by natural evolution. We mutate and recombine a population of candidate solutions (individuals) based on their performance (fitness) in our environment. The individuals that have higher fitness than the average fitness of the population will reproduce and contribute their genetic makeup to future generations. One of the major reasons for using this population-based stochastic algorithm is that it quickly generates individuals that have robust performance. Every individual in the population represents one instantiation of a force law.

Further discussion of the genetic algorithm needs clear definitions of the parameter sets used for the GA individuals. The evolving parameters of the LJ force law are:

- ϵ : strength of the robot-robot interactions.
- c : non-negative attractive robot-robot parameter.
- d : non-negative repulsive robot-robot parameter.
- F_{max} : maximum force of robot-robot interactions.

and similar 4-tuples for obstacle-robot and goal-robot interactions.

Offspring are generated using one-point crossover with a crossover rate of 60%. Mutation adds/subtracts an amount drawn from a N(0, δ) Gaussian distribution. Each parameter has a $1/L$ probability of being mutated, where L is the number of parameters. Mutation ensures that parameter values stay within accepted ranges. Since we are using an EA that minimizes, the performance of an individual is measured as a weighted sum of penalties.

$$Fitness = W_1 * CollPen + W_2 * CohPen + W_3 * GoalNotReachedPen \quad (2)$$

The weighted fitness function consists of three components, a penalty for collisions, a penalty for lack of cohesion, and a penalty for robots not reaching the goal. Since there is no safety zone around the obstacles [1], a penalty is added to the score if the robots collide with obstacles. The cohesion penalty is derived from the fact that in a good hexagonal lattice, interior robots should have six local neighbors. A penalty occurs if a robot has more or less neighbors. If no robot reaches the goal within the time limit, a further penalty occurs.

Experimental Methodology of Offline Learning Module: Our 2D simulation is 900 × 700, and contains a goal, obstacles and robots. Up to a maximum of 100 robots and 100 static obstacles with one static goal are placed in the environment. The goal is always placed at a random position in the right side of the world, while the robots are initialized in the bottom left area. The obstacles are randomly distributed throughout the environment, but are kept 50 units away from the initial location of the robots, to give the robots the opportunity to first get into formation. Each circular obstacle has radius R_0 of 10, and the square shaped goal is 20 × 20.

When 100 obstacles are placed in the environment, roughly 5% of the environment is covered by the obstacles (similar to [1]). The desired separation between robots R is 16, and the maximum velocity V_{max} is 20. Figure 2 shows 40 robots navigating through randomly positioned obstacles. The larger circles are obstacles and the square to the right is the goal. Robots can sense other robots within the distance of $1.5R$, and can sense the obstacles within the distance of R_0+1 (minimum sensing distance). The goal can be sensed at any distance.

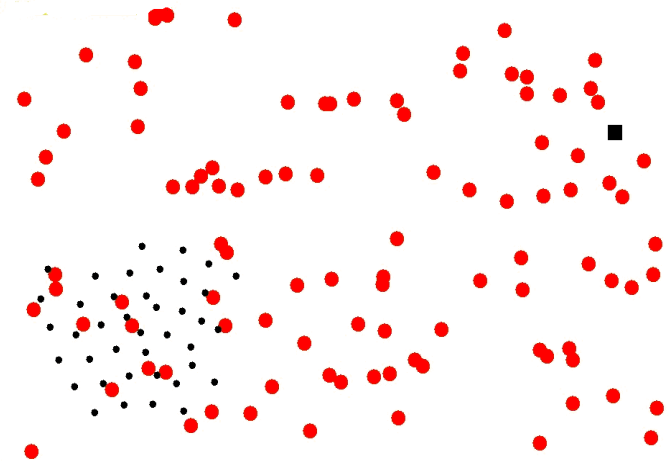


Fig. 2. 40 robots moving to the goal. The larger circle represent obstacles, while the square in the upper right represents the goal.

An LJ force law was evolved using the offline learning module of our simulation tool. The population size was 100 and the EA was run for 100 generations. We trained over scenarios with 40 robots and 90 obstacles. Each individual (an instance of the force law) was evaluated for 1500 time steps, and averaged over 50 random instantiations of the environment.

Constraints with Offline Approach: The offline approach produces successful results [4], assuming that there is a global observer that assigns the fitness to individuals, the robots face scenarios in their current environment similar to

the environment in which they were trained, and there are no communication delays or disruptions. In real world online applications, an offline approach can be problematic due to unexpected environment changes. We need an approach that allows the robots to rapidly adapt to the changing environment. We propose DAEDALUS, a fully distributed online approach that allows robots to adapt to the environment changes.

Online Approach with DAEDALUS: Each robot of the swarm is an individual in a population that interacts with its neighbors. Each robot contains a *slightly mutated* copy of the optimized force law rule set found with offline learning. This ensures that our robots are not completely homogeneous. We allowed this slight heterogeneity because when the environment changes, some mutations perform better than others. The robots that perform well in the environment will have higher fitness than the robots that perform poorly. When low fitness robots encounter high fitness robots, the low fitness robots ask for the high fitness robot's rules. Hence, better performing robots share their knowledge with their poorer performing neighbors.

When we apply DAEDALUS to obstacle avoidance, we focus on two aspects of our swarm: reducing obstacle-robot collisions and maintaining the cohesion of the swarm. Robots are penalized if they collide with obstacles and/or if they leave their neighbors behind. The second scenario arises when the robots are left behind in cul-de-sacs. This causes the cohesion of the formation to be reduced.

Experimental Methodology of Online Adaptation: Each robot of the swarm contains a slightly mutated copy of the optimized LJ force law rule set found with offline learning and all robots have the same fitness at the start. There are five goals to achieve in a long corridor, and between each randomly positioned goal is a different obstacle course with 90 randomly positioned obstacles. The online 2D world is 1650×950, which is larger than the offline world. In our changed environment, each obstacle has a radius of 30 compared to the offline obstacle radius of 10. So more than 16% of the online environment is covered with the obstacles. Compared to the offline environment, the online environment triples the obstacle coverage. We also increase the maximum velocity of the robots to 30 units/sec, making the robots moves 1.5 times faster than in the offline environment. The LJ force law learned in offline mode is not sufficient for this more difficult environment, producing collisions with obstacles (due to the higher velocity), and robots that never reach the goal (due to the high percentage of obstacles).

Robots that are left behind (due to obstacle cul-de-sacs) do not proceed to the next goal, but the robots that had collisions and made it to the goal are allowed to proceed to the next goal. We assume that damaged robots can be repaired once they reach a goal.

Results: To measure the performance of the DAEDALUS approach, an experiment was carried out with 60 robots, 5 goals in the long corridor, and 90 obstacles in between each goal. The experiment was averaged over 50 runs of

different robot, goal, and obstacle placements. Each robot is given equal initial fitness and "seeded" with a mutated copy of the optimized LJ force law learned in offline mode. If a robot collides with an obstacle, it's fitness is reduced. Whenever a robot encounters another robot with higher fitness, it takes the relevant parameters pertaining to the obstacle-robot interaction of the better performing robot.

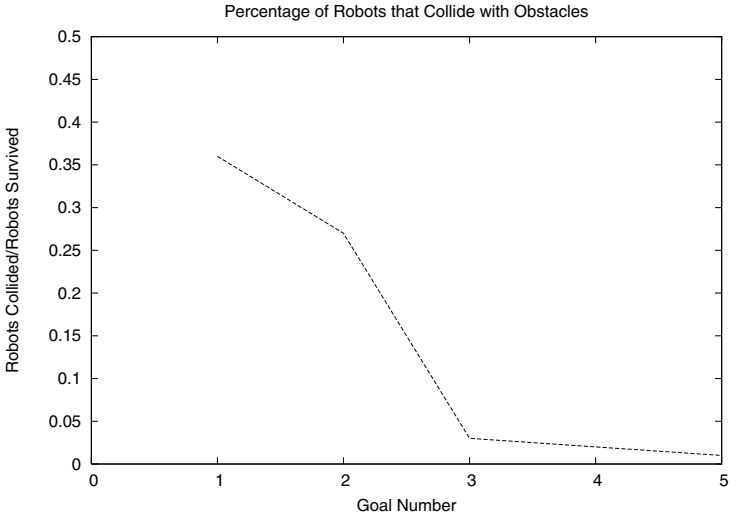


Fig. 3. The ratio of colliding robots versus the number of surviving robots, for 60 robots moving through 5 goals with 90 obstacles in between each goal

Figure 3 shows the ratio of the number of robots that collided with obstacles versus the number of robots that survived to reach the goals. The graph indicates that after only 2 goals, the percentage of robots that collide with obstacles has dropped from about 36% to well under 5%. Inspection of the obstacle-robot parameters indicates that the repulsive component increased through the online process of mutation and the copying of superior force laws (this was confirmed via inspection of the mutated force laws).

This first experiment did not attempt to alleviate the situation where robots are left behind; in fact, only roughly 43% of the original 60 robots reach the final goal (see Figure 4, lower line). This is caused by the large number of cul-de-sacs produced by the large obstacle density. Our second experiment attempts to alleviate this problem by focusing on the robot-robot interactions. Our assumption was that the LJ force law needs to provide stronger cohesion, so that robots aren't left behind.

If robots are stuck behind in cul-de-sacs (i.e. they make no progress towards the goal) and they sense neighbors, they slightly mutate the robot-robot interaction parameters of their force laws. In a situation in which they do not sense the presence of neighbors and do not progress towards the goal, they rapidly

mutate their robot-goal interaction causing a "panic behavior". These relatively large perturbations of the force law allow the robots to escape their motionless state.

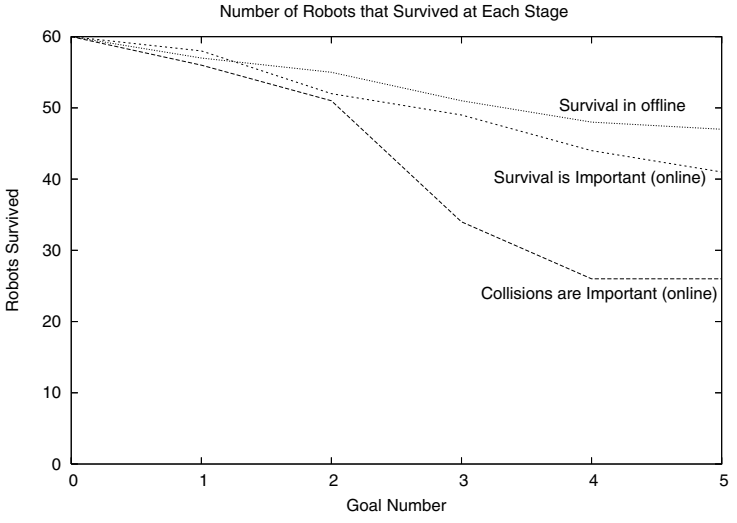


Fig. 4. A comparison of (a) the number of robots that survive when rules are learned using offline learning, (b) the number of robots that survive when using online learning (where the focus is on reducing collisions), and (c) the number of robots that survive when using online learning (and the focus in on survivability)

Figure 4 shows the results of this second experiment. In comparison with the first experiment (with survival rates of 43%), the survival rates have increased to 68%. As a control experiment, we ran our offline approach on this more difficult task. After five goals, the survival rate is about 78%. Recall that the offline results are obtained by running an EA with a population size of 100 for 100 generations, with each individual averaged over 50 random instantiations of the environment. As can be seen, the DAEDALUS approach provides results only somewhat inferior to the offline approach, in real time, while the robots are in the environment.

Although not shown in the graph, it is important to point out that the collision rates were not affected in the second experiment. Hence, we believe that it is quite feasible to combine both aspects in the future. Collision avoidance can be improved via mutation of the obstacle-robot interaction, while survival can be improved via mutation of the robot-robot interaction and robot-goal interaction.

3.2 Evolution of Self Assembling Agents

For our second application we examine the self-assembly of robotic machines. These machines must determine their physical structures in order to consume

some resource and output some product. A self-assembling machine may need to compete for resources and rapidly adapt in a changing environment. In this dynamic simulated environment, machines without an explicitly defined fitness function compete for survival. Each machine is slightly different from the others. Those machines that perform better send their "blue-prints" to those that perform worse. The poorer performing machines rebuild themselves according to the new blue-print. This study uses DAEDALUS to examine the issues in adapting L-Systems [5] to represent the different stages of development of these machines. We present our preliminary results of this on-going study.

In this simulation, a number of machines are placed within a virtual environment, where they must compete for three distinct resources. Acquisition of a sufficient quantity of the available resources determines survival. Two of these resources will be referred to as light and heat, and the third resource is simply space; machines are not allowed to overlap. Machines use the acquired resources as energy for survival and growth.

L-Systems Approach and Experimental Methodology: A machine's final structure is determined by a stochastic context-free bracketed L-system. An L-system is a system of string re-write rules similar to a Chomsky grammar [5]. The L-system is defined as an ordered triplet G = <V, w, P> where V is the alphabet, w is the axiom (or start symbol) and P is the set of production rules. As in any grammar, the start symbol is transformed through derivation steps involving the production rules, resulting in a new word at each step. In the stochastic L-system one or more production rules may have the same start symbol. A probability is attached to each start symbol and the probabilities for a set of production rules with equal start symbols sum to 1. During a derivation step, when a symbol with several rules is encountered a random value is generated to decide which rule to use.

A graphical interpretation of the word is presented at each step. The graphical representation is accomplished through turtle-graphics. Each letter in the alphabet is mapped to a turtle operation. We show an example L-system with the corresponding turtle interpretation of the alphabet used below.
V = {F, B, +, -, [,] }
w = F
P = {F (50%)→ FB, F (50%)→ F[+FB]F}
derivation example:
step 0: F
step 1: FB (at random the first F-rule was chosen)
step 2: F[+FB]FB (at random the second F-rule was chosen)

Turtle interpretation for the alphabet:
F = forward
B = blueprint
+ = turn right
- = turn left
[= push the turtle's current state onto a stack
] = pop the top of the stack into the current turtle

Initially, a set of machines is generated with minor random variations. At each time step machines may collect resources. The world is divided into columns representing the first resource, light. Any machine that is taller than all others in a column collects the light from that column for that time step. The second resource is distributed evenly by mass. Competition for space occurs naturally since machines may not overlap at the base. At the end of each time step all machines are allowed to perform a growth step (L-System derivation), or to switch to an adult stage. The switch to adult stage occurs upon acquisition of a threshold amount of resource. The change arrests the L-System derivations and begins a propagation phase. Adult machines create copies of their genomes (blueprints) and distribute them in the environment. A machine constructor is assumed to exist. The constructor takes blueprints in pairs, recombines them, and builds the resulting machine with the current state of its L-System equal to its axiom.

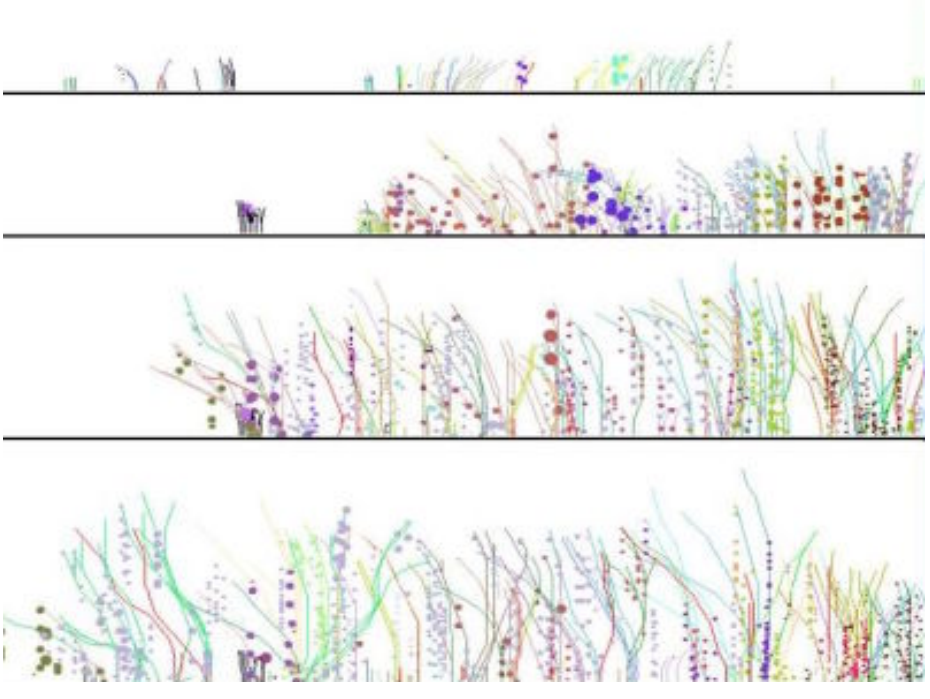


Fig. 5. Progress of Machines Competing for Resources

A series of snapshots from an example run of the simulation is shown in Figure 5. In the first row we see a population of machines shortly after initialization. In the second row the right hand side of the environment is entirely controlled by a dense cluster of tall machines. The left hand side is relatively empty with only one small cluster of machines controlling any resources. The final two rows show the small population being overrun by the much taller machines as they react to

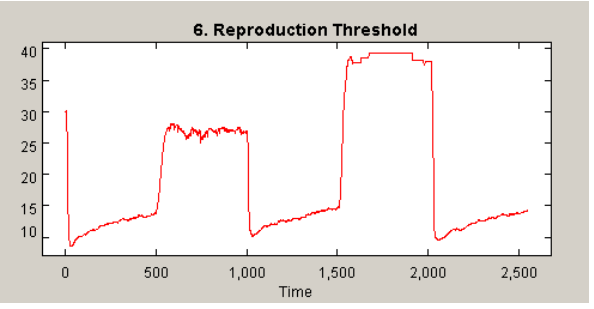


Fig. 6. The reproduction threshold depends on the difficulty in acquiring the resource

the free resource area on the left. Throughout all frames we see the population adjusting its average height and structure for maximal energy absorption.

As mentioned above, the switch to adult stage occurs when the amount of resource acquired exceeds some threshold. We also used DAEDALUS to determine this threshold, as the environment changed. We assume that the machines must expend energy to acquire the resource. In some environments acquiring resources is inexpensive, while in others it is expensive.

Figure 6 illustrates the results. At first, the energy to operate per unit time is 0.1 (the units are arbitrary). Via mutation of the reproduction threshold parameter, machines evolve such that they reproduce when the threshold is approximately 15. There are also a large number of machines (not shown in the graph). After 500 time units the operation energy is 0.5 (it is harder to acquire resources). Note that the threshold increases to roughly 27. There are fewer machines also. The operation energy decreases back to 0.1 and then increases to 0.9. In this case the threshold is very high, at approximately 39. There are also very few machines. Finally, the operation energy decreases back to 0.1.

The results are intuitively pleasing. First, in "easy" areas there are lots of machines that reproduce often. In "hard" areas there are fewer machines that reproduce less often. Second, the results are reproducible in the sense that when the operation energy is 0.1, roughly the same threshold is evolved.

4 Conclusion

Traditional approaches to designing multi-agent systems are offline, and assume the presence of a global observer. However, this approach will not work in real-time online systems. This paper presents a novel approach to solving this problem, called DAEDALUS, where we show how concepts from population genetics can be used with swarms of agents to provide fast online adaptive learning in changing environments. Two case studies are used in this paper to illustrate the feasibility of this approach.

Future work will focus more on the issue of credit assignment. Current work in classifier systems uses mechanisms such as "bucket-brigade" or "profit shar-

ing" to allocate rewards to individual "agents" appropriately [3] . However, these techniques rely on global blackboards and assume that all agents can potentially act with all others, through a bidding process. We intend to modify these approaches so that they are fully distributed, and appropriate for online systems.

References

1. Balch, T., Hybinette, M.: Social Potentials for Scalable Multi-Robot Formations. IEEE International Conference on Robotics and Automation. (2000)
2. Grefenstette, J.: A system for learning control strategies with genetic algorithms. Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann (1989) 183–190
3. Grefenstette, J.: Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms. Springer-Verlag **3** (1988) 225–245
4. Hettiarachchi, S., Spears, W.: Moving Swarm Formations Through Obstacle Fields. International Conference on Artificial Intelligence. CSREA Press **1** (2005) 97–103
5. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer-Verlag. (2004)
6. Spears, W.: Simple Subpopulation Schemes. Proceedings of the Evolutionary Programming Conference. World Scientific (1994) 296–307
7. Spears, W., Spears, D., Hamann, J., Heil, R.: Distributed, Physics-Based Control of Swarm of Vehicles. Autonomous Robots. Kluwer **17** (2004) 137–164
8. Watson, R., Ficici, S., Pollack, J.: Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems. Elsevier **39** (2002) 1–18
9. Wu, A., Schultz, A., Agah, A.: Evolving control for distributed micro air vehicles. Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation. IEEE Press (1999) 174–179

Run-Time Agents as a Means of Reconciling Flexibility and Scalability of Services

Tiziana Margaria[1] and Bernhard Steffen[2]

[1] Universität Göttingen, Lotzestr. 16-18, 37083 Göttingen, Germany
margaria@cs.uni-goettingen.de
[2] Universität Dortmund, Baroper Str. 301, 44221 Dortmund, Germany
steffen@cs.uni-dortmund.de

Abstract. In this paper, we present our approach to flexibly modelling user processes: lightweight run time agents. These agents can be thought of as essentially being 'session handlers with persistent memory, which steer the execution of generic global services according to the users' profile and its current context and goals. We illustrate this paradigm in two application scenarios, the Online Conference Service, and MaTRICS, a remote configuration management environment.

1 Services and Agents in the jABC

The concepts of objects, components, features, and agents are successfully used today in the *Application Building Center* (ABC) environment [15], a tool for graphical, library-based application development. We need them in order to marry the modelling of functionally complex communication systems at the application level with an object-oriented, component based implementation (see Fig.1). Characteristic of the ABC is the coarse-grained approach to modelling and design, which guarantees the scalability to capture large complex systems. The interplay of the different features and components is realized via a coordination-based approach, which is an easily understandable modelling paradigm of system-wide (business-) processes, and thus adequate for the needs of industrial application developers.

As explained in [11], we choose to model applications at coarse granularity in combination with a 'simple' component and coordination model, which support easy component reuse. This design decision was central for the success of the approach, measured in terms of a seamless integration of our technology into the customers' processes.

In this paper, we consider a scenario of increasingly complex, distributed, and interdependent service provision, with the emphasis on our approach of lightweight run time agents. The challenge in such a scenario is combining the apparently inconsistent needs for

– *flexibility*, in terms of complete online reconfigurability of the features, of the whole service, of the role definition, down to personalization issues, and

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 257–268, 2006.

– *scalability*, necessary when managing a number of such services in parallel on the same ASP cluster, each with its own incarnation of the features, services, and roles, and this with
– *telecommunication-level quality of service*, concerning primarily service availability and reliability.

In particular, service interruption, e.g. for reconfiguration and restart, causes a severe damage to the service provider. Thus everything in a service and in its environment must be re-definable at runtime.

Our approach addresses scenarios where the *user-level flow of control* is of central importance and therefore adequately realized via coordination graphs. Particularly well-suited for our approach are applications where the flow of control frequently needs adaptation or updating, as it is the case when addressing user-specific workflows or situation-specific (business-) processes.

The jABC uses lightweight runtime agents (cf. Sec. 2) to individually model user support. According to the rich collection of definitions of agents in [3], they behave like agents in the sense of e.g. AIMA and Hayes-Roth. Other definitions, e.g. by IBM, where "An IBM Agent carries out some set of operations on behalf of a user or another program, i.e., is a task-specific agent", seem to require being software in order to be an agent: these definitions apply to the coarse-grained description of the behavior (as a Service Logic Graph) together with the software that implement it and its basic components.

In our opinion, both points of view are justifiable. Also the authors' own definition of [3] is applicable in our case:

> *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*

Technically, the agents developed within the ABC can be classified as

– **residential**, in the sense that the service logic resides on a single machine (more precisely on a load-balanced cluster, which is however transparent to the SLG and therefore to the agent), which are
– **proactive**, in the sense that the agent *knows autonomously* which features are enabled at this moment for his user, which contents are available for him, and which consequences has a user action wrt. the further navigation potential within this session, and
– **online adaptive**, in the sense that they react instantly to changes in the context (user preferences, configurations of role and rights, other actions by other agents). E.g., a report checked in by the last reviewer may automatically open a discussion forum for this paper available for the whole PC.

The concrete application scenarios considered in the paper are the Online Conference Service ([7,9,8,6]), the conference management system that has been used for the submissions to WRAC, and the MaTRICS ([2]), an agent-enabled platform for the remote intelligent management of computer systems (e.g. server farms).

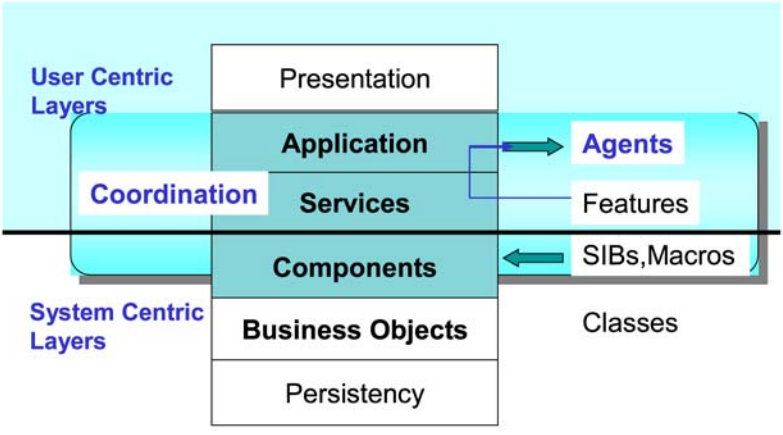


Fig. 1. The Layered Architecture of the ABC Applications

The ABC is used here for the definition and enforcement of complex behaviors in terms of *features*, [4,6][1] which are *superposed* on and coordinated within the system under development. The challenge is precisely how to handle this superposition and coordination in an understandable, well partitioned, and manageable way.

In the following, we first introduce jABC's lightweight runtime agents in Sect. 2, before we present the Online Conference Service (OCS) in Sect. 3, explain our behavior-based agent construction in Sect. 4, and present our feature-based system development in Sec. 5. Finally, we illustrate our approach using the MaTRICS application in Sect.6, and present our conclusions in Sect. 7.

2 Lightweight Runtime Agents

We are convinced that agents are a convenient metaphor for understanding the complex and intertwined behaviors arising in systems like the OCS and MaTRICS. Indeed, from the point of view of an application designer, the jABC is an Application Building Center, but for our business-oriented customers and for end users the same ABC is considered an *Agent Building Center*. End users, and business or product developers that take the stake of the end users, have a strongly *subjective* viewpoint of the system's functionality, of the sort "What does the system do for me?", or more precisely "What does the system do on my behalf?". This mindset is well captured by an interpretation of the behavior in terms of an intelligent and proactive *user's representative* that runs through the system on behalf of the real human user.

[1] Although we too learned to know and appreciate the concept and the use of features in the context of Intelligent Networks [4,5,16], our notion of features is more general in order to also capture a more general class of services like online, financial, monitoring, reporting, and intelligence services.

We call this representative a *Lightweight Runtime Agent*, and we implement it as a projection on the global behavior of the system according to the capabilities, roles, and situative contexts of the user.

In the ABC, we use coordination graphs [11] as a basis for the definition of these projective agents. Together, they make services (expressed as interoperating collections of *features*) available at runtime to the single users. Coordination graphs are called in the ABC Service Logic Graphs (SLGs) in analogy to the terminology for Intelligent Network services. They provide an adequate abstraction mechanism to be used in conceptual modelling: they direct developers to the identification and promotion of user-level interactions as first-class entities of the application development, which is important for taming the complexity of system construction and evolution.

3 A jABC Application: The Online Conference Service

The OCS (Online Conference Service) (see [7] for a description of the service and of its method of development) pro-actively helps authors, Program Committee chairs, Program Committee members, and reviewers to cooperate efficiently during their collaborative handling of the composition of a conference program. It is customizable and flexibly reconfigurable online at any time for each role, for each conference, and for each user. The OCS has been successfully used for over 60 computer science conferences, and many of the ETAPS Conferences. This year it served in addition to conferences also several ETAPS workshops, with 8 instances of the service running in parallel.

The services capabilities are grouped in *features*, which are assigned to specific *roles*. In the OCS, a single user may cover many roles (e.g., PC Members may submit papers and thus be simultaneously Authors), and can switch between them at any time during a working session. A fine granular roles and rights management system takes care of the adequate administration of the context, role and user-specific permissions and restrictions. The different roles cooperate during the lifetime of a PC's operations and use the OCS capabilities, which are provisioned at the feature level. Through the cooperation of its features, the OCS provides a timely, transparent, and secure handling of the papers and of the related submission, review, report and decision management tasks.

The SLG of the OCS has currently approximately 2200 nodes and 3200 edges. The benefit of our hierarchical graphical modelling becomes apparent, when considering the top-most global, application-level SLG (see Fig. 2, top right), which is quite simple:

- it contains at the top level the logic of the *backbone service*, which is basically a skeleton service providing generic internet login and session management services, and
- it coordinates the calls to and the interferences between the single *features*.

In Fig.1, the central layers for understanding our use of agents are the services and the application layer.

Services Layer
The Services layer defines the palette of coarse-grained services offered to the users, and implements them in terms of the available components. In the ABC at the service level we explicitly speak of *Features*, like the Setup, Delegation, and Articles management features, which are accessible in the OCS to (some of the) end-users. Features are explicitly made externally available via the GUI.

Application Layer
The Application layer defines the coarse structure of the workflows of the various potential users in their specific roles in terms of a global SLG that coordinates the involved features. E.g., in Fig. 2 we see on the top right the overall SLG of the OCS, and on the left the SLG for the Submit Article feature.

In particular, an SLG defines user/situation-specific *agents* via

- workflows that are enabled only at *certain times* (e.g. the submission of reports makes sense only between certain deadlines), or
- for *certain roles* (e.g. the configuration of the whole OCS service for a conference, is in the hands of the PC Chairs), or
- under *certain circumstances* (e.g. the update of a task list for a PC Member follows a delegation act of a paper to this member, the discussion in an online forum is enabled only during the discussion phase, only if the PC Chair has opened this forum, and only if the PC Member is not banned via conflicts from accessing this particular forum).

4 From Global Behavior to Personal Agents

The SLG corresponds to the Global Functional Plane in advanced telecommunication services: it describes the universe of the *possible* behaviors in a very abstract way.

At runtime, a lightweight personal *agent* is associated to each user and for each session: the agent can be seen as in charge of navigating the Application level Service Logic Graph for this user in the current context, i.e., taking appropriate care of the current user profile and of behavior's limitations due to roles, deadlines, conflicts, enabled or disabled features, history etc...

Typical agent implementations do not have a layer corresponding to the SLG: they usually consist of a number of agents (programs) that provide functionalities and/or impersonate roles. Their global coordination is traditionally only implicitly defined, and de facto seen as a runtime matter. In order to "see" a certain global behavior, one has either to simulate the parallel behavior of the cohort of agents, or to construct the corresponding model and analyze it, which is unfortunately a classical source of state explosion problems.

In our opinion, this is a clear drawback: due to the application profile we are treating, we need to be able to analyze the global behavior at a more abstract level: the graphs of hundreds of users in parallel would quickly exceed memory limit. In contrast to other component-based approaches, e.g., for object-oriented

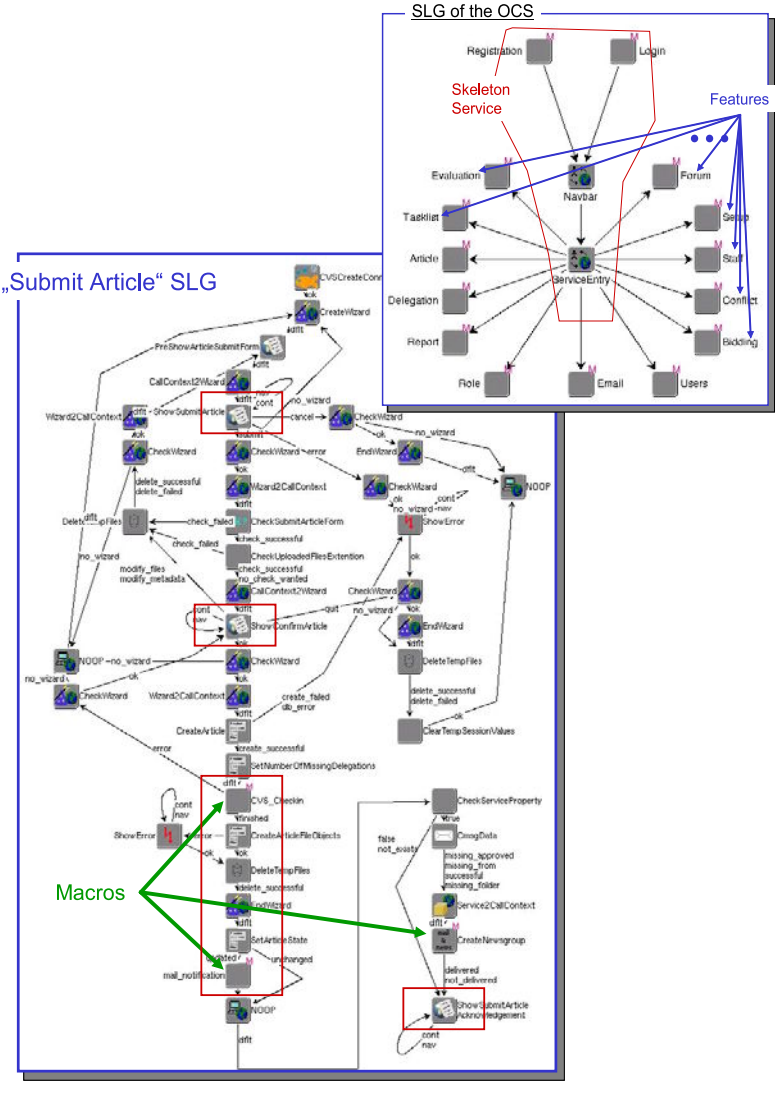


Fig. 2. A Hierarchical Macro: The Submit Article Feature

program development, the ABC focusses on the dynamic behavior: (complex) functionalities are graphically stuck together to yield flow graph-like structures (the SLGs) embodying the application behavior in terms of control, as shown in Fig. 2 for a portion of the OCS. These SLGs constitute therefore directly our coordination models, and they are at the same time coarse-grained formal models of the business logic of the applications. As explained in [11], they are internally modelled as labelled Kripke Transition Systems [12] whose nodes represent (elementary) SIBs and whose edges represent branching conditions, so that we can directly model check the SLGs as they are.

Additionally, SLGs can be automatically analyzed, for example via Model Checking. However, this feature is not in the focus of this paper and interested readers can refer to [11,10] in order to read about the formal-methods based capabilities of the ABC.

The ABC contains an iterative model checker based on the techniques of [14]: it is optimized for dealing with the large numbers of constraints which are characteristic for our approach, in order to allow verification in real time. In fact, libraries of constraints allow the model checker to individually check hundreds of typically very small and application- and purpose-specific constraints over the flow graph structure. This provides concise and comprehensible diagnostic information in case of a constraint violation: in fact, the feedback is provided directly on the SLG, i.e. at the application level rather than on the code.

Our lightweight runtime agents have the same effect from a user's point of view as traditional software agents, but enable a more efficient and powerful analysis (e.g. of feature interactions, of safety and liveness properties, of deadlocks) on this additional SLG-based modelling level.

The following properties, which address important security and confidentiality precautions, exemplify, what can be automatically checked by our model checker:

- the service can be accessed only by registered users,
- users can freely register only for the role Author,
- the roles Reviewer, PC Member, PC Chair are sensitive, and conferred to users by the administrator only,
- users in sensitive roles are granted well-defined access rights to paper information,
- users in sensitive roles agree to treat all data they access within the service as confidential.

These properties are instances of the classes of safety and consistency requirements identified in [1] to be characteristic of Computer Supported Collaborative Work platforms. They are specific instantiations of Role-Based Access Control (RBAC) models [13]. Being able to automatically verify such properties via model checking is a clear advantage of the ABC, and it is essential in order to guarantee the safety of the kind of applications we build.

5 Feature Based System Development and Lightweight Runtime Agents

Since features are not independent but rather influence each other, one of the central aims of service validation via model checking and testing is exactly the discovery and control of the so called *feature interactions*. Since features are realized in the ABC in terms of (possibly nested) macros, each with its own SLG, also in this case we can directly work with this model.

In the ABC, features are enabled and published to the end-users on their finer granularity, according to a complex, personalized role-right-context management. As an example, only users with a PC Chair role are able to submit

articles in the name of another user. The design of the sub-structure of features is driven exactly by the needs of *distinguishing behaviors* according to different *contexts*. Sub-features in fact usually arise by refinement of features as a consequence of the refinement of the configuration features and of the role-rights management system. This way we enable a very precise fine-tuning of the access to sensitive information and to protected actions.

Once an internet service is online, it is continuously navigated in parallel by a cohort of agents that are 'proxies' for the single users. They "represent" the user, i.e., execute its global service logic within the limits imposed to the user they are associated with.

Accordingly, in the ABC we do not directly program the agents or their knowledge base. Rather, the SLG of an application defines the space of potential behaviors that agents can assume, and each agent's behavior is defined implicitly as the currently valid projection onto this potential, filtered via

1. the roles-and-rights management system, which defines dynamic, reconfigurable projections on the behaviors defined in the SLG, and
2. the current global status of the application, including the data space, the configuration, and certain event- and time-dependent permissions.

6 Application: Agent-Based Remote Configuration of Systems in the MaTRICS

With MaTRICS, we describe an architecture for *pervasive management* of distributed systems. It allows remotely connected users (e.g., system administrators) to modify the configuration of any service provided by a specific (application) server, like email-, news- or web-servers. Novel to our approach is that the system can manage configuration processes on heterogeneous software- and hardware- platforms, which are performed from a variety of peripherals unmatched in today's practice. In this sense, we realize a sort of *pervasive system management*, where devices like mobile phones, faxes, PDAs are enabled to be used by system managers as remote system configuration and management tools.

A physical server system usually provides a number of different services, each coming with an own configuration format that depends on the version numbers of the services and the underlying operating systems, and which are stored in different files distributed over the file system. Updating a service thus results in adapting the old configuration to the new one, which requires the execution of a set of activities, expressible by means of a *configuration workflow*.

6.1 The Architecture

In the MaTRICS, every configuration workflow for a service is defined by an operational *model* that expresses its business logic, including alternatives, special cases, and exception handling. This model, also expressed as Service Logic Graph, is formally defined and automatically analyzable (e.g., via model checking). Configuration workflows are executed in the ConfigManager, the centralized core component of the MaTRICS.

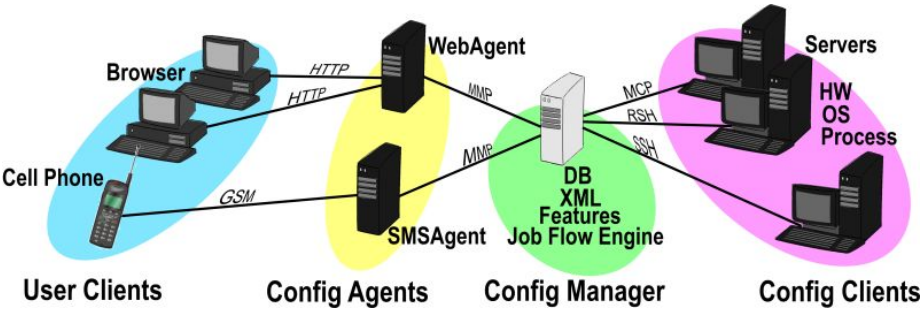


Fig. 3. The High-level Architecture of MaTRICS

As shown in Fig. 3, *Users* access the MaTRICS from their thin terminals via protocol-specific agents called ConfigAgents.

The *ConfigAgents* serve as protocol adapters between the client-specific protocols and the ConfigManager. They realize the presentation of data for a user client and transform the user requests into a uniform format for the ConfigManager. To support a variety of different communication systems, like Web, SMS, EMail, FAX, etc., we need specialized ConfigAgents, which we also realize by means of SLGs.

The servers to be configured, called *ConfigClients*, communicate with the ConfigManager via standard protocols, like SSH or RSH. All the modifications of a ConfigClient's configuration are launched by the ConfigAgents on behalf of a user requesting the execution of some configuration workflows on that client. The workflow execution is carried out by the *ConfigManager* via a JobFlow-Engine.

The SLG modelling style is powerful and intuitive, and it is used to define all the user-level configuration services: an example of it is the WebAgent in Fig. 4, but all the workflow-oriented services within the MaTRICS core like the ConfigClient synchronization workflow and the service for editing configfiles are designed as SLGs.

Already this global intuitive visualization is an important advantage wrt. to the number of scripts and manual commands of which the workflow implementation for server management typically consists: it is well known how difficult it is getting familiar with a large number of complex system-level tasks!

6.2 The MaTRICS ConfigAgents

The ConfigAgents are lightweight agents that constitute the communication interface for the user clients. They realize the presentation of data for a user client, transform the user requests into a uniform format for the ConfigManager, and act on behalf of that user to launch reconfiguration processes. To support a variety of remote lighweight communication systems, like Web-browser, SMS, EMail, FAX, etc., the MaTRICS provides specialized ConfigAgents that are themselves realized by means of SLGs, in an abstract yet intuitive modelling style.

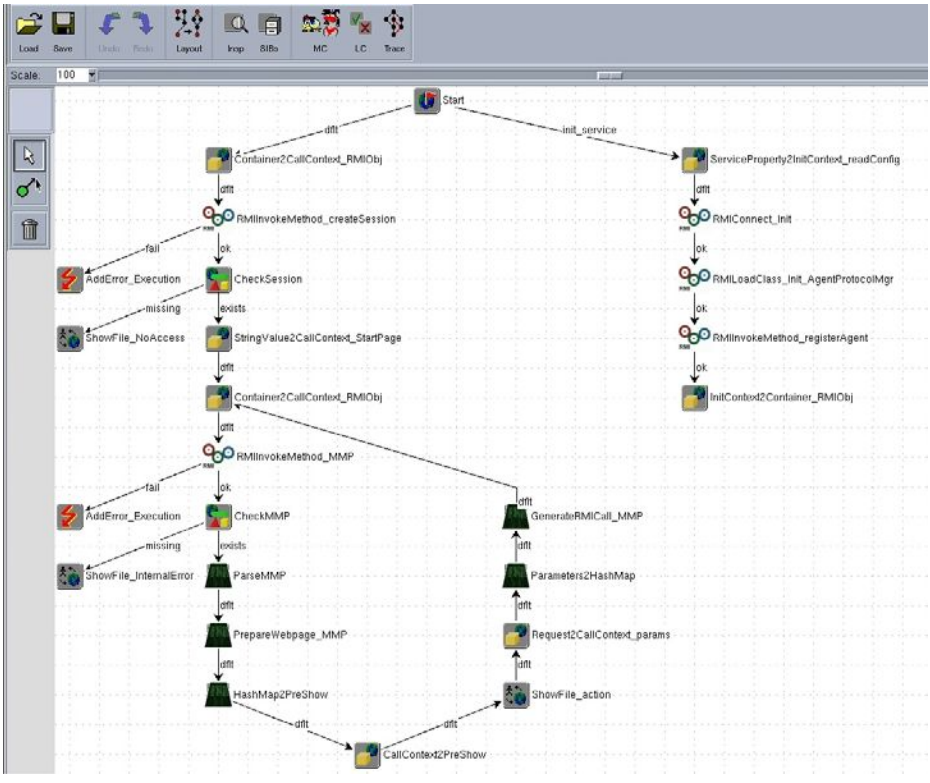


Fig. 4. The Service Logic Graph of the MaTRICS' WebAgent

We consider now the example of the WebAgent, a specialized agent which supports reconfigurations via a browser user interface. Its SLG, shown in Fig. 4, is organized as follows: from the start node,

- the *init_service* branch leads to the portion of the logic that initializes the WebAgent's RMI-interface. To this aim, it connects with the ConfigManager and requests a unique agent identifier.
- The first time a user client connects to the WebAgent, the dflt branch is followed and a user session is created in the ConfigManager (RMIInvokeMethod-Create_Session).
- As soon as the entry point of the navigation service of the ConfigManager is entered, that navigation service takes control over the WebAgent and constructs the navigation bar. The corresponding requests and responses to the ConfigManager are encapsulated by the MaTRICS Manager Protocol (MMP), which internally uses RMI for communication. At this point the user is able to use the WebAgent to carry out one or several remote operations.
- For each operation, the SIB RMIInvokeMethod_MMP encapsulates the user request as MMP message and invokes a configuration service of the Con-

figManager. The response must be transformed into a web page, which is shown by the SIB ShowFile_action. Any modification on this page by the user client is read and encapsulated to MMP, then transmitted by the SIB RMIInvokeMethod_MMP. At this point, the loop is closed and the WebAgent is ready to carry out another operation.

7 Conclusions

We have presented lightweight run time agents, our approach to flexibly modelling user processes. In contrast to the classical understanding of agents, lightweight run time agents have no code on their own, but are merely 'session handlers with persistent memory, which steer the execution of generic global services according to the users' profile and its current context and goals. The advantage of this approach is that vital properties of the intended service can be statically verified at the global process level on the corresponding SLG, which essentially summarizes the behavior of all its individual lightweight run time agents. Moreover, abstracting from implementation details like distribution of execution, data bases, and other platform-specific issues, lightweight run time agents provide an intuitive user-level paradigm.

We are currently using our approach in a variety of industrial projects, in order to evaluate our claims. These projects are very illuminating, and already led to some modifications enhancing the practicality of our approach, like a new kind of process-oriented documentation on the basis of the SLG, and various tailored versions of concurrency.

Acknowledgements

Many thanks are due to Markus Bajohr and Martin Karusseit for their support and help in discussions on the MaTRICS and the OCS.

References

1. T. Ahmed, A. Tripathi: *Static Verification of Security Requirements in Role Based CSCW Systems*, Proc. 8th Symp. on Access Control Models and Technologies, Como (I), ACM Press, pp.196-203, 2003.
2. M. Bajohr, T. Margaria: *MaTRICS: A Management Tool for Remote Intelligent Configuration of (Pervasive) Systems*, Proc. ICPS 2005, IEEE Int. Conference on Pervasive Services 11-14.July 2005, Santorini, Greece, IEEE Computer Society Press.
3. S. Franklin, A. Graesser: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, Proc. 3rd Int. Worksh. on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996. http://www.msci.memphis.edu/~franklin/AgentProg.html
4. ITU: *General recommendations on telephone switching and signaling intelligent network: Introduction to intelligent network capability set 1*, Recommendation Q.1211, Telecommunication Standardization Sector of ITU, Geneva, Mar. 1993.

5. ITU-T: *Recommendation Q.1203. "Intelligent Network - Global Functional Plane Architecture"*, Oct. 1992.
6. M. Karusseit, T. Margaria: *Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service*, WWV'05. 1st Int'l Workshop on Automated Specification and Verification of Web Sites, Valencia, Spain, March 14-15, 2005, – Post Workshop Proc. appear in ENTCS.
7. B. Lindner, T. Margaria, B. Steffen: *Ein personalisierter Internetdienst fr wissenschaftliche Begutachtungsprozesse*, GI-VOI-BITKOM-OCG-TeleTrusT Konferenz Elektronische Geschäfts-prozesse (eBusiness Processes), Universität Klagenfurt, September 2001, http://syssec.uni-klu.ac.at/EBP2001/ .
8. Tiziana Margaria: *Components, Features, and Agents in the ABC*. In Objects, Agents, and Features, Revised and Invited Papers from the International Seminar on Objects, Agents, and Features, Dagstuhl Castle, Germany, February 2003, LNCS 2975, pp. 154-174, Springer Verlag, 2003
9. T. Margaria, M. Karusseit: *Community Usage of the Online Conference Service: an Experience Report from three CS Conferences*, 2nd IFIP Conf. on "e-commerce, e-business, e-government" (I3E 2002), Lisboa (P), Oct. 2002, in "Towards the Knowledge Society", Kluwer, pp.497-511.
10. T. Margaria, R. Nagel, B. Steffen: *Remote Integration and Coordination of Verification Tools in jETI*, Proc. ECBS 2005, 12th IEEE Int. Conf. on the Engineering of Computer Based Systems April 2005, Greenbelt (USA).
11. T. Margaria B. Steffen: *Lightweight Coarse-grained Coordination: A Scalable System-Level Approach*, in STTT, Special Section on Formal Methods in Industrial Critical Systems of the Int. Journal on Software Tools for Technology Transfer Vol.5, N.2-3, 2004, Springer Verlag, pp.107-123.
12. M. Müller-Olm, D. Schmidt, B. Steffen: *Model-Checking: A Tutorial Introduction*, Proc. SAS'99, September 1999, LNCS 1503, pp. 330–354, Springer Verlag.
13. R. Sandhu, E. Coyne, H. Feinstein, C. Youman: *Role-Based Access Control Models*, IEEE Computer, 29(2):38-47, Feb. 1996.
14. B. Steffen, A. Claßen, M. Klein, J. Knoop, T. Margaria: *The Fixpoint Analysis Machine*, (invited paper) CONCUR'95, Pittsburgh (USA), August 1995, LNCS 962, Springer Verlag.
15. B. Steffen, T. Margaria: *METAFrame in Practice: Intelligent Network Service Design*, In *Correct System Design – Issues, Methods and Perspectives*, LNCS 1710, Springer Verlag, 1999, pp. 390-415.
16. B. Steffen, T. Margaria, V. Braun, N. Kalt: Hierarchical service definition, Annual Review of Communication, Int. Engin. Consortium (IEC), 1997, pp. 847-856.
17. H. Weber: *Continuous Engineering of Information and Communication Infrastructures*, Proc. Int. Conf. on Fundamental Approaches to Software Engineering (FASE'99), Amsterdam, LNCS N. 1577, Springer Verlag, pp. 22-29.

Concept and Sensor Network Approach to Computing: The Lexicon Acquisition Component

Jan Smid[1], Marek Obitko[2], and Andrej Bencur[3]

[1] SKS Enterprises, Finksburg, MD 21048, USA
jsmid@sks99.com
[2] Department of Cybernetics, Czech Technical University, Prague, Czech Republic
obitko@labe.felk.cvut.cz
[3] Department of Measurement and Control, VSB – Technical University Ostrava, Czech Republic
andrej.bencur@vsb.cz

Abstract. In this paper, we describe an on-going project called Concept and Sensor Networks (CSN). The development of this project has been described and discussed in past PSMP workshops [1]. The purpose of the project is to develop a framework for entities that can process sensor information into concepts. One of the features of this proposed network is the ability for the entities to use language communication to exchange concepts. These entities can potentially represent concepts, knowledge and information using different kinds of semantics. To further this project, we will to implement the proposed framework using physical and virtual sensors. In this paper, we overview the key components of the project, primarily focusing on lexical acquisition and the corresponding algorithm.

1 Introduction

Concept networks represent knowledge and information. They consist of communicating nodes of different power, which monitor and control physical environments, collect environmental data, infer meaning, build lexicon, and communicate this information. The specific concepts within the network need to be communicated and updated based on information provided by agents and sensors. Some of our efforts in this area are summarized in [2]. Many of the aspects of these networks have been captured during the PSMP (Knowledge Presentation Sharing, Mining and Protection) workshops [1].

CSN project is important because it introduces new definitions about networks and addresses the question of how to build a system that possesses the six main features introduced in section 2 (universal data communication, universal semantic communication, system tolerance, recursive action semantics, lexicon evolution, learning by sharing). To our knowledge there is no such theory available in the literature.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 269 – 280, 2006.

Two main terms we are dealing with are concepts and sensors. There is no universally accepted definition of the concept. Concepts are usually defined respective to an application. We will work with any heuristic definition of the concept that reflects a compact description of the world in the sense expressed by Baum [16]. Concepts can be rules, programs, a set of regression coefficients, or a subset of object defined by specific attributes.

Even a trivial domain such as a number representing a sensor measurement leads to a nontrivially complicated semantics space. That is why we would like to prove our ideas using the simplest domains. The working assumption is that complexity does not depend primarily on the complexity of the application domain.

Sensors can be virtual or physical. An example of a physical sensor is a piece of wire. An example of a virtual sensor is a query counter. Our view of sensors and nodes is radically different from the centralized approach in which a few sensors and nodes are directly connected to a powerful computing unit. Sensor networks consist of small, low-cost sensors, typically with wireless connection and with limited capabilities and operating range. One of the virtues of these networks is their ability to accept new nodes. Another benefit is fault tolerance and the distributed processing of the data. These networks can be modeled using the standard multi-agent systems paradigm. The agent can be a simple unit carrying a sensor that reports the sensed value. It can also be a more powerful unit that is able to analyze data from multiple sensors.

A critical issue for sensor networks is the communication between nodes. There are two aspect of communication: data transport and information semantics. The routing protocols in a dynamically changing topology are already available, for example in Intel-Berkeley motes [5], also standard sensor networks are summarized by Akyildiz [3]. On the semantic side, the current approaches require pre-programmed and fixed way of exchanging information. We believe that in order to ensure truly autonomous and fault-tolerant systems, the agents must be able to acquire knowledge and skills from other agents and humans.

The rest of this paper is organized as follows: First, we summarize the overall project goals, the project principles and summary. We discuss entities used in sensor networks and discuss the hardware layer. The core of the paper is the explanation of the lexicon acquisition and its advantages. Here we start with explaining our view of the role of the language and discuss its usage, particularly the correspondence between utterances and semantics, together with inference. Then we describe in detail our algorithmic layer of lexicon acquisition, including applications and implementations. We conclude with the overview of the future work.

2 Overall Project Goals and Principles

Our project consists of four layers - cognitive models, message transport models and implementations, hardware and software embedding, and applications. The long-term plan follows several important steps that have been discussed under the umbrella of the PSMP workshops [1].

2.2 System Main Principles

The main system principles are:

1. Universal data communication

A number of protocols can be used to transmit data. Some of these protocols are universal. We also use wireless connections in our networks. The most common protocol for basic data transmission is the serial protocol. This protocol is often inconvenient because it may be very based on an application and the designer of the application. The Hypertext Transfer Protocol (HTTP) of the Internet stack is more appropriate for our purposes because it satisfies the universality requirement. Most agents and human user will be able to process the data due to this universal feature.

2. Universal semantic communication

Even if distributed systems are built around different knowledge representations they should be able to incorporate new information from different agents and share it. The information communicated should have in general two components, the description (utterance), and the semantics. The interpretation of utterances is a notoriously hard problem for standard syntactically based linguistics. Language with emphasized semantic components seems to be closer to methods that humans use well.

3. System tolerance

A flexible system needs to be able to incorporate new members and remove existing members. We will refer to this feature as system tolerance.

4. Recursive action semantics

In general, most researchers use semantics to refer to the meaning of an utterance, We add to this definition by incorporating the ability to perform actions on a given item. An example is the merge of two actions.

5. Lexicon and semantics

The system is able to build and utilize the lexicon. The lexicon builder is based on an algorithm operating on pairs of descriptions (utterances) and semantics (rules, programs, objects, actions).

6. Learning by Sharing and Dialog

Learning can be accomplished in a number of different ways. Solving problems, mathematical and game puzzles, searching for new theorems and mathematical proofs was the traditional focus in artificial intelligence. Unfortunately, there are very few of us that can claim to have discovered a novel theorem within out lifetime. A more realistic avenue for learning is by sharing information. We advocate learning by primarily sharing as a predecessor of the highest intellectual abilities.

2.3 Project Summary

The project goals can be summarized as follows:

1. Information sharing

We want to develop a framework, based on the two universal principles (introduced above), that allows the system to accept without manual reprogramming information that is either solicited or given.

2. Semantics
Every piece of information has syntactical and semantic component which can often have the similar structure. Fundamental operations over pairs of syntactical-semantic components allow using language for communication between entities.

3. Applications
The purpose of the whole system is to perform useful tasks, such as providing information sharing, information merge and object monitoring without repeated re-programming.

Current applications in process range from a simple temperature reading to a world of agents that are able to build temperature profiles and use them to recognize and inform about abnormal.

3 Entities Used in Sensor Networks

We distinguish between passive agents, which merely report data, stand-alone agents, which are able to process data on a lower layer, and language-based agents, which are able to communicate about information and exchange semantics. We define

websensors = sensors + (micro)processor + http(tcp/ip) interface,
webprocessors = http(tcp/ip)interface + processor

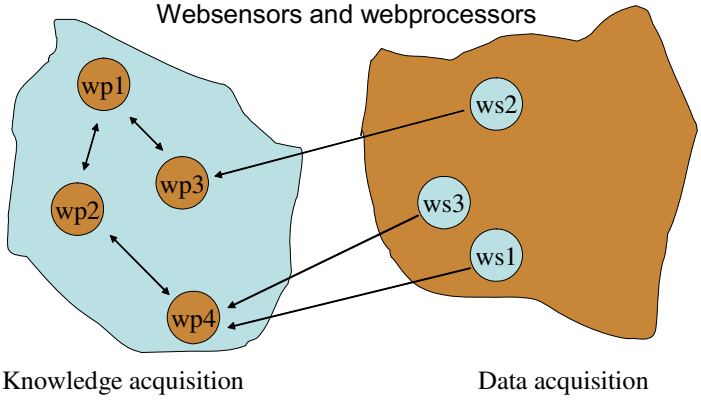


Fig. 1. Websensors and Webprocessors

The purpose of the hardware layer is to form a network that communicates using universal protocols, such as TCP/IP and HTTP. There are several different types of processors and interfaces we have used for our experiments. One group consists of simple processors such as Basic Stamp2 and Javelin, both from Parallax [6] or the HC08 by Freescale. These processors communicate using mini-web servers such as Red-i and Siteplayer [7] that provide HTTP connections. The other group of

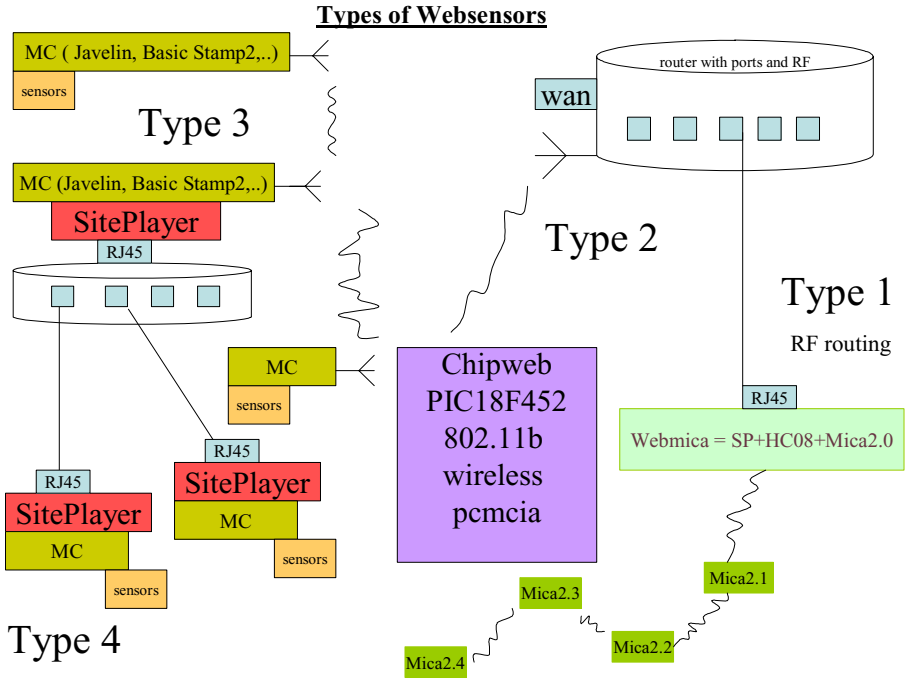


Fig. 2. Types of websensors

processors consists of integrated sensors, micro-controllers and transmitters. One outstanding instance of this group is the mica mote by Intel Berkeley [5] and distributed by Crossbow. For the overall picture, see figure 2.

3.1 Webmica

We have modified mica motes and created an example of a websensor (see figure 3). The interface for standard mica2 is provided by a custom made board consisting of generic micro-controllers. For practical reasons a Freescale HC08 microcontroller was programmed and it is able to interface of-the-shelf webserver Site-Player distributed by NetMedia [7].

The mica wireless nodes redirect data to a designated mica2 node that serves as a base of the network. The base outputs serial data that are converted into a command for SitePlayer by additional HC08 processor. A webpage is created by a SitePlayer module. For example a command for writing byte 0x55 to the memory location 0x20 reads:

(0x80, 0x20, 0x50)
(write-command, address-to-write-to, data-to-be-written).

Freescale's MC68HC908KX8 8-bit microcontroller has been chosen for our application, but any other device with similar features could be used as well. One of

our reasons for this choice was the minimal number of external components necessary for running this controller. Typical quartz crystal does not have to be put on the board, because of internal oscillator, which can be easily trimmed for exact frequency to generate baud rate for the serial communication module (UART).

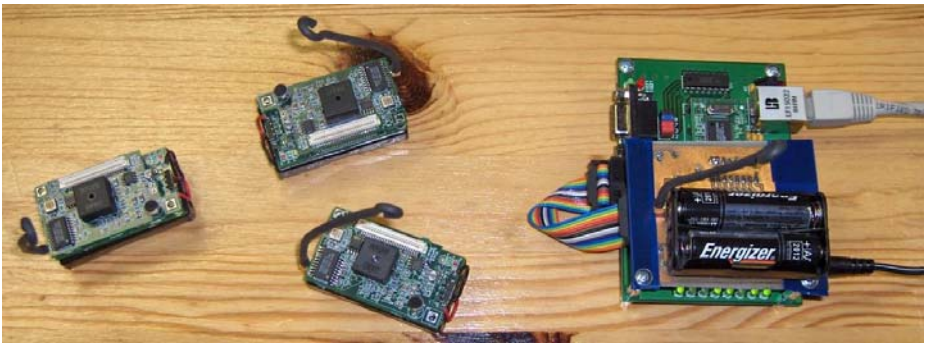


Fig. 3. Webmica and mica2 nodes cluster

The main task of this processor is to wait for a new packet to arrive from the wireless sensor network's node 0 (MICA2) and then to delete certain bytes carrying data based on the number and order of the bytes, such as Node ID, Packet number. Any sensor can be connected to MICA sensor board, which gives many possibilities of future expansions, including object monitoring with infrared sensors, ultrasound sensors, and smoke detectors.

4 The Role of Language

Semantics has the central role in our approach to CSN. Language has several functions from the semantics point of view.

1. Language utterance can point to actions or to declarative statements
2. Language provides convenient and critical shortcuts decreasing complexity
3. Language is context depending and hierarchical

String languages have dominated mathematics and computer science at least since the beginning of the 20th century. The formal definition of the Thue word game was presented by a Norwegian mathematician Alex Thue around 1914. The words in this game are equivalent if there is a finite sequence of substitutions, from a given dictionary, taking one word into other. This game was as an example of an unsolvable problem by a computer. E. Mark Gold showed in 1967 [10] that it in some cases is impossible to identify even a simple regular language. In our opinion these examples provide fascinating results about syntax but they do not address primary issues of communication and the role of language. These types of problems and syntax analysis have prevailed in computer science because of its usefulness for computer programming languages. Following Winograd and Flores [11], p.124, we can

paraphrase their statement as “Our critique and approach is not a condemnation of the technical work that has been done or even of the specific techniques (representations, deductive logic, frames, meta-description, etc.) that have been developed. It challenges the common understanding of how these techniques are related to the human use of language”.

5 Semantics, Inference and Utterances

Jackson [14] discusses “semantic information” carried by a sentence as simply the data structure that the program creates when it processes (“understands”) the sentence. The general *problem of semantics* is to discover and define the kinds of “semantics” data structure represents.

In addition to the defining the semantic data structure we also need to address semantic inference and semantic retrieval. Using the analogy of natural creatures, the system can learn independently of language. Modules performing useful functions can be controlled by logic provided by some other modules. When a response of the system is ready, regardless of how it was obtained, it needs to be communicated using language. This leads to the problem of generating utterances from semantics.

These procedures can be best outlined using the simplest database semantics and inference. One of the simplest models which incorporates logic is Datalog (a version of Prolog for databases). We use Datalog as a paradigm for implementation and as an illustration for traditional concept of semantics.

Standard database semantics (e.g. Ullman [11]) can be represented by first-order predicate logic. There are three alternative ways to define “meaning” or semantics. Regardless of which definition is used, the inference procedure of the system leads to a generation of a new frame, object or semantics and generating a corresponding utterance. Even in the simple case of declarative semantic there is no guarantee that a unique answer is defined, or that there is a reasonable way to turn the declarative program into a sequence of steps that compute the answer (Ullman [11]). The alternative ways of defining semantics are

1. Proof-theoretic interpretation of rules is the set of all the facts that can be proved in all possible ways.
2. Model-theoretic interpretation of rules is a collection of predicates assigned the truth or falsehood to every possible instance of those predicates. Usually, an interpretation is represented by its set of true instances.
3. Computational definition of meaning provides an execution algorithm to tell whether a potential fact (predicate with constants for its arguments) is true or false.

Standard semantics, used by e.g. Poole [15] for modeling robots is a kind of database type of semantics described by Ullman.

Recursive action semantics: we believe that the standard definition and declarative rules must be expanded by using procedure, programs and recursive operations, or functions that operate on themselves. This means that, for examp, an object is always bundled with other objects and functions that operate on them. Semantics becomes a

way and a domain how systems "thinks" and a special case of that are rule, standard logic and thinking using language or words.

Let us consider knowledge as a pair (utterance, semantics) acquired by a node. Simple semantics is particularly associated with declarative facts. One type of simplest processing is to apply standard logic operation for declarative facts. Typical logic operations that could be arguably innate or provided by the system creator (and used by ordinary humans) are modus ponens and modus tollens. Let us emphasize that the system is designed to accept any pair of the type (utterance, semantics). This pair can represent a rule, a declarative fact, or essentially anything. A special case of semantic databases are pictures (in e.g. jpeg format) that are annotated.

The retrieval step is based on both the current utterance and the semantics. The simplest task is analyzing an utterance which points to a unique semantic situation. This situation can be translated using a known lexicon. The next more general step calls for a semantic situation that is found based on a given utterance and a current or preferable semantics.

6 Algorithmic Layer-Lexicon Acquisition

Our approach emphasizes that knowledge representation and acquisition tasks should be based on *flexible* language communication. The central idea is that virtual agents communicate using evolving lexicon and language. In the traditional context-free setting, language acquisition and identification, is a difficult task that cannot be successfully accomplished by using only positive examples [10]. Traditional artificial intelligence approaches difficult problems as the responsibility of an individual agent. We redefine an agent's task as the ability to accept knowledge and skills provided by other agents and humans without strong logical reasoning capabilities. We can observe that most humans learn the majority of their capabilities from other people. These learned concepts are sometimes slightly adapted by some people and applied for ever-changing environments and conditions. Only in very exceptional cases we are required to create a completely novel solution. Consequently we believe we should "start small" with agents allowing them to learn from other agents and develop skills to tackle completely novel situations.

We have attempted to apply a more human-inspired method of knowledge acquisition to our network. One of the main features of our artificial system is that information and knowledge for agents is provided in the form of pairs (utterance-semantic observation). Siskind [8] proposes that an utterance can be paired to its semantics. The semantic component of the pair can be expressed as a program or a semantic expression. The mapping between utterances and semantics needs to be discovered by the agents. In doing so, agents are able to learn to understand new utterances, and moreover are able to acquire the skills, since they can run newly acquired programs by themselves. This approach emphasizes distributed nature of agent systems. Our approach is consistent with speech acts and the semantic orientation in the sense that our framework supports any pair (utterance, semantics).

We describe a simple algorithm that allows us to expand an agent's lexicon. The methods Siskind used in [8] were based on the arc consistency algorithm [9], standard

backtracking can work as well however we use what we call the semantics matching algorithm. First, an agent is presented several utterances and their semantics (expressed in a symbolic or actual programming language):

Utterance	Symbolic Utterance Translation - Semantics
foo bar	LIST temperature
baz bar	LIST humidity
quux bar	LIST acceleration
baz bleen quux plugh	COPY humidity TO acceleration
baz bleen cruft foo	COPY humidity FROM temperature

The arc consistency algorithm can be visualized as a graph. The nodes show variables *V1 foo*, *V2 bar*, … *V7 cruft*, and their possible values (e.g. only one value *temperature* for the variable *V1 foo*) that are filtered out using the domain consistency rule and the semantic matching. The links connect overlapping sets of possible values. The link numbers give the order in which they are deleted when applying the Arc Consistency [9] algorithm.

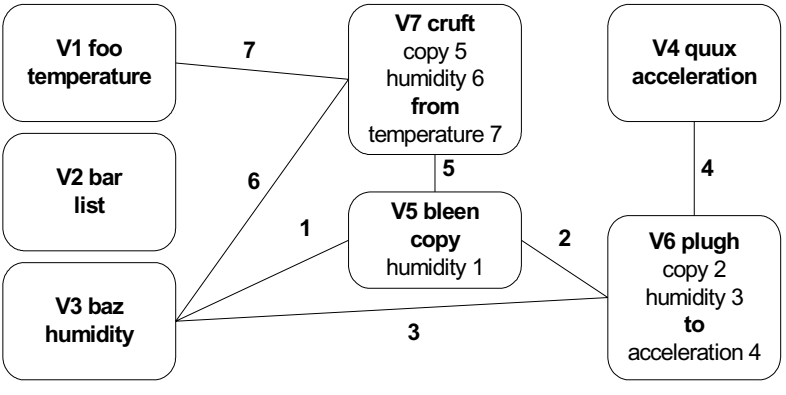


Fig. 4. Result of the Arc Consistency algorithm

This algorithm provides the following lexicon.

Term	Translation
bar	LIST
baz	humidity
bleen	COPY
cruft	FROM

Term	Translation
foo	temperature
quux	acceleration
plugh	TO

The "translation" refers to elements in a programming language. Thus, whole utterances can then be translated into an executable program, which can be performed by agents as a response to the utterance. The algorithmic layer of this translation game involves incorporating several tasks which correspond to additional more algorithms.

These include effective information representation and search using such ideas as the small world paradigm, semantics incorporation and reasoning in the semantic layer.

Semantics matching is much simpler and faster for the semantic mapping problem. Let us illustrate the semantic matching algorithm using an example of two utterances. Two utterances and corresponding semantic expressions are

foo bar = list(temperature)
baz bar = list(humidity)

The "*list*" is the only item that shows twice and therefore the "*bar*" corresponds to the "*list*". That implies that *foo = temperature* and *baz = humidity* and this case is solved. Similarly we can proceed for pairs involving more items. The *(m, n)* refers now to a case where one utterance consists of m items and the other of n.

- Case *(2, 2)* *(a, b)* *(c, b)* implies *b* is known
- Case *(3, 2)* *(a, b, c)* *(b, d)* implies *b* is known
- Case *(3, 3)* *(a, b, c)* *(b, c)* implies *a* is resolved, *b*, *c* domains of values are reduced
- Case *(4, 2)* *(a,b,c,d)* *(b,c,f,e)* implies *b*, *c*, *d* are reduced to 2 semantic expression values each
- Case *(4, 3)* *(a,b,c,d)* *(a,b,c,e)* implies *a*, *b*, *c* are reduced to 3 semantic values each

By induction it can be shown that any *(m, n)* situation can be reduced to *(m-1, n)*, *(m, n-1)* or better.

8 Implementations

We have implemented several hardware components and software components of the CSN system. The hardware components, such as webmica were already described earlier. In this section, we focus on the implementation of the software components.

The web interface accepting pairs is implemented in PHP and can be accessed via web browser by a human or by any agent via HTTP. The interface allows us to view, enter and edit semantic pairs. The semantics of these pairs can be immediately evaluated.

In our implementation, we use PHP and SQL, which can be both easily exchanged and executed by agents without any need for the compilation step or a special programming context. A script example corresponding to an utterance "*tell temperature*" is

```
<?php  require_once("csn.php");
       require_once("semantics.php");
       PrintValue("tempovsd");            ?>
```

The `csn.php` provides the basic database connectivity. The `semantics.php` contains hard coded functions that express the basic semantics used in utterances. For example, the `PrintValue` function is defined (using `csn.php`) as

```
function PrintValue($what) {
  print getSingleValueViaSQL(
          "SELECT $what FROM sensors LIMIT $max,1"); }
```

which means selection and printing the last measured value from the SQL table. The function `getSingleValueViaSQL()` provides basic MySQL support function that allows to extract required information in this environment. The mapping from the utterance "*tell temperature*" to "*PrintValue('tempovsd')*" is then obvious – *tell* maps to *PrintValue*, while *temperature* maps to *'tempovsd'*. This mapping is predicated on the selection of the correct semantic frame and we do not discuss it in this paper.

It is important to notice that when agents are able to exchange and execute these scripts then they are able to perform these operations themselves. Thus, after learning the lexicon and correspondence to the programs, they acquired the other agent's skills.

The semantic mapping algorithm is implemented in Java. The component accepts pairs of utterances and semantics, and is able to find mappings between them. For each pair, both utterance and semantics is tokenized. The tokens are used in the mapping graph. The mapping graph is built incrementally as new pairs arrive. It represents the graph of possible mappings (i.e. where utterance and semantics occur in at least one pair) minus the graph of impossible mappings (i.e. where utterance and semantics do not occur in at least one pair). We do not permit uncertainties in mappings so far – the mapping can be only strict 1:1, without any noise in both utterances and semantics. Using the algorithm described above, when a mapping is exact (i.e. one to one), it is removed. This procedure is repeated for each mapping that becomes exact during the process of lexicon and semantics acquisition. In the end, an utterance can be mapped to one element of semantics (procedure/function or its parameter) or to nothing (when the utterance doesn't have impact on semantics). This mapping is then used for translation from utterance to semantics or vice versa.

9 Conclusion and Future Work

We have presented both hardware and software components for the CSN project. These basic components support information and concept exchange and algorithms for lexicon acquisition. The current implementations are mainly in PHP, Java, and MySQL because of the simplicity of the implementation. We expect the communication component to be implemented using the transport layer protocol TCP/IP and HTTP, to provide extensible agent communication.

With regard to immediate future work, there are many components of the project that need to be refined or implemented. The next practical application that we intend to implement is database merge and intelligent object monitoring. These applications consist of two or more webprocessors and associated websensors (figures 1 and 2). One processor archives data and knowledge from its sensors. There is a need to merge this knowledge with a similar archive of data and concepts from the other webprocessor. More generally, more webprocessors and websensors (small hardware boxes) can be distributed as independent groups. When these groups contact each other, they will have the capability to communicate using the acquired lexicon resulting in new profiles, reasoning rules and sets of programs.

Second application deals with a task of building a model of temperature (or other quantity) in a spatial distribution. Some agents might know how to build these models

and can share this skill with other agents that are needed to building the complete temperature model.

CSN project is important because it addresses the question of how to build a system that possesses the six main features introduced in section 2 (universal data communication, universal semantic communication, system tolerance, recursive action semantics, lexicon evolution, learning by sharing). To our knowledge this problem has not been address in the computer science literature so far. Implemented CSN will be useful for home monitoring by anyone (ease of use) and for professional monitoring because of the flexibility of the system.

Acknowledgements

Jan Smid was supported in part by the CIBAC grant,Bowie State University. Andrej Bencur was supported in part by the grant GACR 102/05/H525, TU Ostrava.

References

1. PSMP (Knowledge Presentation Sharing, Mining and Protection) Workshop Series Website. http://sks99.com/psmp.php (2003-2005)
2. Bencur, A., Pokorny, J., Smid, J.: Communication of Autonomous Systems Based On Wireless Sensor Motes. The 7th WSEAS. Prague (2005)
3. Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. A Survey on Sensor Networks. IEEE Communications Magazine (2002)
4. J. Smid, M. Obitko, V. Snasel. Semantically Based Knowledge Representation. Communications in Computing (2004)
5. Crossbow Technology Inc. Website. http://www.xbow.com/
6. Paralax Website. http://www.parallax.com/
7. Siteplayer Website. http://siteplayer.com/
8. Siskind, J.: Learning Word-to-meaning Mapping. In Broeder, P., Murre, J. (eds): Models of Language Acquisition. Oxford University Press (2000)
9. Mackworth, A. K.: Constraint Satisfaction. In Shapiro, S. (ed.): Encyclopedia of Artificial Intelligence. John Wiley 1987, pp. 205-211.
10. Gold, E. M.: Language Identification in the Limit. Information and Control. 10. (1967) 447-474
11. Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Addison-Wesley (1988)
12. Dale, R., Moisl, H., Somers , H. (eds): Handbook of Natural Language Processing
13. Ullman, J.D.: Principles of Database and Knowledge-based Systems. Vol. I and II. Computer Science Press (1988)
14. Jackson, P. C.: Introduction to Artificial Intelligence. Second Enlarged Edition. Dover Publications (1984)
15. Poole, D., Mackworth, A. K., Goelble, R.: Computational Intelligence: A logical Approach. Oxford University Press (1998)
16. Baum, E. B.: What Is Thought? The MIT Press (2004)

An Agent Based Hybrid Analog-Digital Robotic Sensor Web Meta-system

A. William Stoffel

NASA's Goddard Space Flight Center
A.W.Stoffel@nasa.gov

Abstract. A low cost, simple, agent based, robotic sensor web to collect exploration data is a major and efficient asset to any exploration initiative. The Hybrid Analog-Digital Robotic Sensor Web (HADRS) provides this asset. HADRS is a meta-system rather than a system because it is an operational concept. This is the case, as HADRS can be used on any robotic system, of any physical configuration. HADRS will also operate in any environment, terrestrial, planetary, and space... HADRS will disperse a large number of inexpensive Mobil Agents or Mobile Units (M/U) over a wide area. Their initial search patterns will be random. The cost is low due to an inexpensive analog subsystem. When the payload instruments detect a target of interest, an autonomous, simple onboard feedback loop between the instrument and the hybrid analog/digital steering system, will guide the robot to the target.

1 Goals

- Maximize the amount of exploration data collected.
- Minimize data loss due to any robotic system's malfunctions.
- Minimize the risk of total, system failure.
- Minimize the size, weight, and power requirements of each Mobile Unit (M/U).
- Minimize any robotic, agent based system costs.

2 HADRS Concept Overview

A low cost, simple, agent based, robotic sensor web to collect exploration data is a major and efficient asset to any exploration initiative. The Hybrid Analog-Digital Robotic Sensor Web (HADRS) provides this asset. HADRS is an operational concept. HADRS can be used on any robotic system, of any physical configuration. HADRS will also operate in any environment, terrestrial, planetary, cyberspace and space...HADRS will disperse a large number of inexpensive Mobil Agents or Mobile Units (M/U) over a wide area. Their initial search patterns will be random. The cost is low due to an inexpensive analog subsystem. When the payload instruments detect a target of interest an autonomous feedback loop between the instrument and the hybrid analog/digital steering system, will guide the robot to the target. Figure 1 is provided as a graphic overview of the concept, for the reader to refer to while reading this paper.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 281 – 285, 2006.

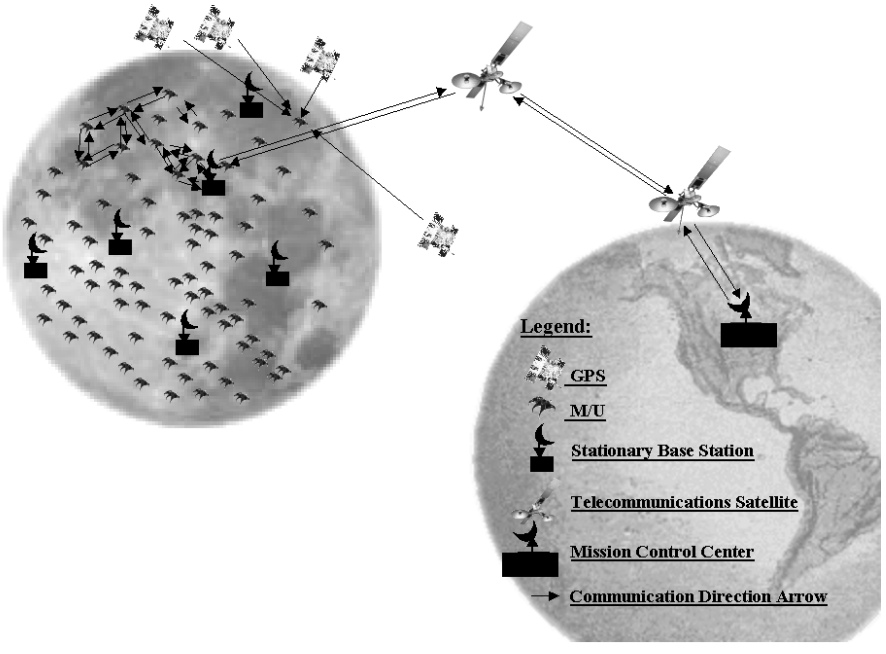


Fig. 1. HADRS Concept Overview

3 HADRS Concept Structure and Function

HADRS acts as the ops concept or meta-system for any agent based system. HADRS is applicable to any environment and a wide range of exploration tasks. It is a low cost, high return, solution to a complex problem. The technology for initial concept development and implementation is currently available. The HADRS sensor web will have near constant contact with telecommunications and other satellites in orbit, planned by other endeavors of the exploration Initiative, and to the Mission Control Center.

HADRS Consists of Four major components:

- The HADRS Sensor Web (S/WB),
- The HADRS Mobile Units (M/U).
- The HADRS Stationary Base Stations.
- The HADRS Mission Control Center.

3.1 HADRS Sensor Web (S/WB)

Each HADRS M/U broadcasts the data and location of where the data was collected, to the nearest members of the sensor web, in other words to other M/Us. This occurs in the same way members of social insect colonies spread information, about the location of food, to other members of their colony. The insects that found the food in

turn spread the information on to other insects and eventually the information gets back to the hive. The same principle operates here. The data is migrated through the sensor web like waves that ripple out from a rock dropped into a pond, until it eventually reaches one or more of the Stationary Base Stations. This accomplishes two things. One, each M/U uses low power and a small bandwidth to transmit information to the rest of the web. Two, it ensures that if any single M/U suffers a fatal failure the mission will not suffer a major loss of data, even if that robot's data did not get to a Stationary Base Station. Other HADRS M/Us will know where the M/U was when it failed, allowing Investigators to make a decision about whether or not to re-acquire the data with other M/Us. The reacquisition will take place after the cause of the first M/U's failure is diagnosed and a comfort level has been attained that the subsequent M/Us will not suffer the same fate. Diagnosis of an M/U's failure will be easier than current remote failure diagnostic techniques, as there will be other M/Us to autonomously assist in the diagnosis, without depending solely on Mission Control's interpretation of historical data.

The HADRS sensor web will have near constant communication with telecommunications and other satellites in orbit, planned by other endeavors of the Exploration Initiative (EI), or directly to the Mission Control Center. The Base Station uplinks data from HADRS to the Mission Control Center and downlinks commands from the Mission Control Center, to the Base Stations. The Base Stations relay it to the whole sensor web The HADRS Base Stations have a large data storage capacity and a powerful communications system, compared to the HADRS M/Us. Since the M/Us have very limited storage capacity, it will be critical that the M/Us be able to move data to the Base Stations on a continuous basis.

3.2 HADRS Mobile Unit (M/U)

Like the rest of the HADRS meta-system, the HADRS M/Us are applicable to any physical configuration of mobile agent, in any environment, using any propulsion method and for any exploration tasks. HADRS does not care what the propulsion method and appendages for the mobility platform are. The type of mobile platform will be determined by the environment in which the M/U is to operate and the task(s) to be accomplished. The size and weight of each HADRS M/U will be minimized by the maximum use of MEMS and Nanotechnology.

Each HADRS Mobile Unit (M/U) will consist of the following components:

- A Hybrid Analog/Digital mobility platform.
- An Exploration oriented Sensor/Instrument package.
- A limited data storage capacity.
- A communications package.
- A power supply.

The main advantage to the HADRS M/U is that the use of analog circuits for most of the initial "What's out there" searching task, are less expensive by several orders of magnitude than conventional digital robots, such as the MERS... There is no intelligence needed for the analog side of the mobility platform as it is designed to conduct the initial search in a random pattern. This part of the HADRS M/U is much like the analog robots built at Los Alamos National Laboratory (Hasslacher and Tilden 1995). This cost reduction allows one to have many HADRS M/Us in the

HADRS Sensor Web for a fraction of the cost of existing agent based robotic systems. The M/Us will initially be disbursed evenly over a large area and then search randomly until they identify something of interest, this gives HADRS the ability to cover a large area, in a short period of time, allowing for and promoting serendipitous discoveries.

The digital part of the HADRS M/U is used to steer the M/U in a particular direction once on-board instrument sensors have detected a target of interest. This is done, autonomously, through a feedback loop from the activated sensor on the instrument package to the steering system. One possible sensor is an airborne particle detector with sufficient on board analysis capability to allow the HADRS M/U to determine what the detected particles are, for example the "electronic nose" from Cyrano Sciences (Albert et. al., 2000). Once particles of interest are detected, the M/U can be guided toward the source of the particles by simply signaling the analog/digital steering system to move the robot in that direction. Once the M/U begins to move in a certain direction the analog system can be set to go to the source of the signaled input. This is a behavior that analog chips can handle easily because you can give them a sense of inertia just by switching the rhythm of the analog chip's signal firing pattern that controls the appendage movements so the chip no longer fires in an offset pattern. The offset pattern is what gives the M/U its random motion. If the sensor then senses that the particle density is getting lower in the direction the robot is going, it signals the digital steering system again. This time the M/U is told to let the analog system move back toward the target until the particle density increases again. At that point, the M/U is, again, told to head in the direction of greatest particle density producing an autonomous feedback loop to steer the M/U. The result is that the system uses very little intelligence but gets to the target autonomously, and in an efficient manner. An ant behaves in much the same manner when trying to stay on a food scent trail. Ants exhibit similar search pattern when it loses the scent trail until it finds it again. This simple feedback loop is the way living

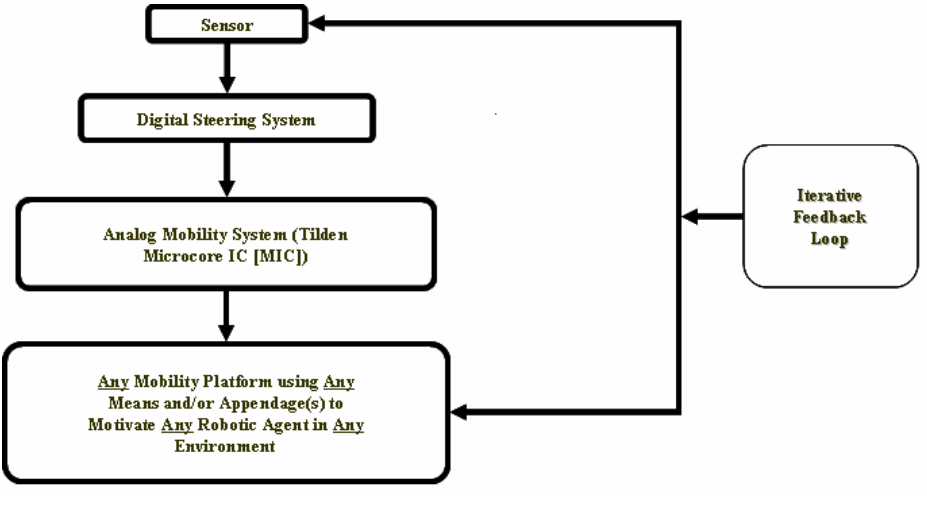


Fig. 2. HADRS Mobile Unit (M/U) Steering System Feedback Loop

organisms perform these tasks, why they can move so efficiently, and with minimal involvement from higher cognitive centers. Below is a simple overview graphic of the HADRS M/U mobility steering system feedback loop (Figure 2).

3.3 The HADRS Stationary Base Stations

The Base Stations can store large amounts of data and have powerful communications equipment. The Base Station platforms are also mobile in order to move to optimal locations, in order to maintain contact with as many M/Us as possible, mission control, and other EI systems. When the optimal position is attained, the base stations are static and no longer mobile. The HADRS Base Stations have a large amount of storage capacity, compared to the HADRS M/Us. Since the M /Us have very limited storage capacity, it will be critical that the HADRS system be able to move data to the Base Stations on a continuous basis. The Stationary Base Stations will be equipped with a powerful communications system to uplink the data to mission control and downlink commands from the Mission Control Center, to the S/WB.

4 Conclusions

HADRS will be a major asset to the exploration initiative. It will be very robust, by reducing any single point of failure. It will be able to explore a large area in a short time, allowing for and promoting serendipitous discoveries. Finally, HADRS is a low cost system for three reasons. One, HADRS will be a system of systems, because it will use other EI systems to complete its mission. Two, it uses smart operational strategies. Three, HADRS minimizes the cost of each HADRS M/U through its innovative design.

References

Hasslacher, B., Tilden, M. W., "Living Machines", ROBOTICS AND AUTONOMOUS SYSTEMS: The Biology and Technology of Intelligent Autonomous Agents. Editor: L. Steels. Elsivier Publishers, Spring 1995. (LAUR-94-2636). http://www.nis.lanl.gov/projects/robot/

Albert, K. J. Lewis N. S., Schauer C. L., Sotzing G. A., Stitzel S. E., Vaid T. P., and. Walt D. R, “Cross-Reactive Chemical Sensor Arrays,” *Chem. Rev*., 2595-2626, (2000).

Harnessing Agent-Based Games Research for Analysis of Collective Agent Behavior in Critical Settings

Abdennour El Rhalibi and A. Taleb-Bendiab

School of Computing and Mathematical Sciences
Liverpool John Moores University
a.elrhalibi@livjm.ac.uk,cmsatale@livjm.ac.uk

Abstract. This paper presents an AI architecture that has been developed specifically for controlling complex multi-agents interaction in games. The model is based on previous research into Emotional Societies and presents a realistic and believable environment for games. In response to a perceived lack of depth and realism in the team relationship dynamics of modern gaming, we developed a human agent architecture, multi-agent system, and demonstrative game application. The agent architecture was based partially on research into social psychology, and utilized emotion and belief representations to drive action selection. Agent interaction and relationship development was produced on the basis of the Iterated Prisoner's Dilemma (IPD), through which a team's success came to be determined by its members' choices to cooperate or compete with its leader. A produced game application illustrated the operation of the developed architecture within the context of a political street protest. A set of evaluation scenarios were devised to test the success of the project work within this game application, and ultimately found it to be successful in achieving a good level of realistic team-based reasoning and interaction. Beside the potential application of the model and architecture to a computer entertainment environment, the model is generic and can be used as well for "serious" application which involves distributed emerging behavior, scenarios based simulation, complex agent-based modeling including emotional, reactive and deliberative reasoning.

1 Introduction

A common witticism concerning teamwork is that there is no 'I' in team—successfully working as a team requires that its individual members put their own interests aside and commit to each other. While if the idea is most certainly cliché, portrayals of teamwork and leadership roles in computer gaming remains noticeably self-centered. While a number of recent games have featured team or squad-based gameplay, few have challenged players as leaders in a realistic and dynamic way. The emphasis remains on the player as the only member of the team that really matters, as well as the only one really capable of surmounting the challenges ahead. The primary element lacking from such games are both the interpersonal aspects of team work situations and the evolving nature of a team relationship, both among its members and with its leader.

This project sought to take some initial steps towards modeling team relationships and leadership roles more realistically in gaming applications. An oft-referenced

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 286 – 298, 2006.

presentation given by Clark Gibson and John O'Brien at the 2001 Game Developer's Conference describes team AI as the, "next step in game AI," and that it is, "needed to be competitive," and, "increase realism," [1]. While game developers within the industry may be content to build this team AI from scratch, a significant body of research is already available from the agent-based academic AI community. The project work sought to harness the results of such research for this purpose. Its goal was to devise an agent architecture and multiagent system to model the chaotic, evolving nature of teamwork dynamics through representations of emotions, beliefs, and relationships. Previous research efforts into human AI, rule-based system theory, and even the Iterated Prisoner's Dilemma were all incorporated into its development. The successful operation of the produced system was illustrated in the context of a custom-made game application involving a team of protestors staging a political demonstration. The ultimate goal of the project work was not only to make initial steps into realizing better teamwork dynamics, but also to rectify the, "mutual lack of interest between serious game developers and academic AI researchers," by highlighting the suitability of agent-based approaches to AI in gaming [2].

The main outcomes of the work are presented in the remaining of the paper.

2 Literature Review

Foundational research for the project was conducted within the broad topic areas of social psychology and human agent AI. In addition, current efforts to model team behavior within the games industry were reviewed and accounted for. Some of the major findings produced by the research are presented in this section.

Social psychology was quickly identified as a primary resource for understanding teamwork and leadership dynamics. A huge body of literature exists concerning how to be an effective leader or manage a successful team, but it was deemed to be too high-level for this initial AI development. At the same time, more specialized modern research into team behavior was thought to be too specific and nuanced for successful use. The best alternative was found in some of the core theories of social psychology—the branch of human psychology dedicated to understanding how human beings behave in response to each other. Among the topics and theory selected as appropriate for use in the project were conformity, social facilitation and loafing, group norms and cohesiveness, group polarization, and the various factors that influence individual cooperation in group settings. Some of the findings validated common conceptions of teamwork—larger groups elicit more powerful pressures to conform, more cohesive groups generally perform better, and individual behavior relative to a group is partly determined by personality, past experience, and situational factors. Other findings provided some interesting theoretical backing to the project work. For example, the presence of a single defector in a group decision-making process has been found to reduce the pressure of conformity on the other members by 80 percent [3]. Another finding deemed to be interesting was that group cohesiveness, or the strength of a team relationship, can be a negative factor in group decision-making as it can influence members to avoid disagreement and neglect their better judgment in order to maintain group compliance [3]. Research into social psychology also aided in the identification of social dilemmas such as the Prisoner's Dilemma, which was later chosen for use in modeling the project's team member interaction.

Perhaps the most important body of research surveyed for the project work was that of human AI—the efforts of a relatively small branch of the academic AI community to devise more and more realistic simulations of human behavior. The topic area was approached through the examination of three exemplary projects developed for applications in computer animation, gaming, and sociological research simulations. Each of the projects will be briefly reviewed here.

For computer animation the selected project was a knowledge-based human crowd animation system developed by S.R. Musse and D. Thalmann in 1997 [4].

The Oz Project was surveyed for an understanding of the state of human AI pursued for applications in computer gaming. The Oz Project's principal 'Tok' agent architecture is pictured in Figure 1. Among the most notable details garnered from its study were its use of a planning system to select appropriate agent actions given both emotional and sensory stimulus, its greater inventory of emotions (including hope, fear, pride, and admiration), and its use of value constructs like standards and attitudes to give its agents individual personality and elicit continual influences on behavior [5].

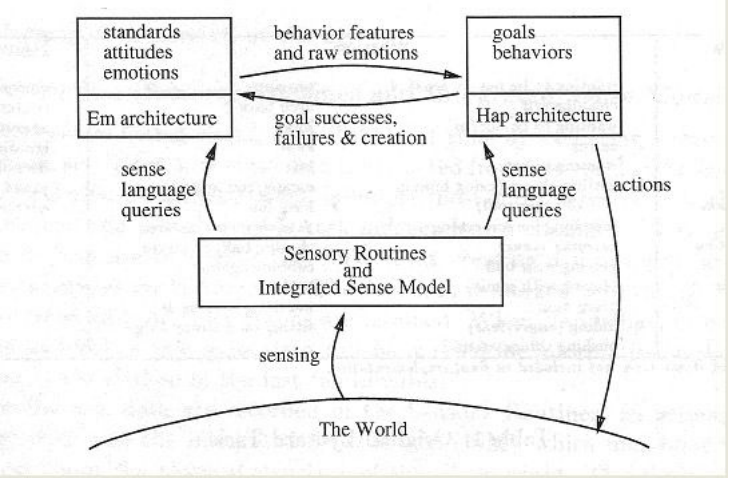


Fig. 1. The Oz Project's 'Tok' Agent Architecture [5][12]

The final and arguably most complex of the reviewed projects was that of the Ackoff Center for Advancement of Systems Approaches (ACASA) in developing an integrative agent architecture and multiagent system for purposes in both gaming and social research. The research group's architecture expanded on the Oz Project's dual subsystem model with the use of four subcomponents, designed to give their agents specialized physiological, emotive, cognitive, and motor/expressive reasoning in an interconnected framework. This particular arrangement influenced the project work in its own divided architecture and introduced the importance of stress and physiology on human AI. One particularly interesting aspect of the ACASA's work involved the use of detailed 'concern ontologies' to specify the moral values of a given agent and guide its emotional deliberation—a mechanism the researchers intended to be complex enough to model existing literature such as, "the ten commandments or the Koran for a moral code," or, "military doctrine for action guidance," [6]. The ACASA

similarly demonstrated the operation of their architecture through the use of a street protest simulation environment.

In addition to surveying academic AI resources for simulating human behavior the project reviewed the current state of the team AI within the games industry to provide context for its work. The two primary methods highlighting through this research were centralized and decentralized approaches. In the former, game AI developers manage team behavior through the use of a rigid command hierarchy in which orders flow downward from a group commander to their subordinates and environmental information flows upward [7]. Though strategically sufficient and arguably realistic in modeling teamwork situations in the military, this first method is poorly suited to more dynamic team representations due to its reliance on strict obedience and homogeneous agents. Decentralized approaches, the second primary method, involve an opposite arrangement in which, "interactions between squad members, rather than a single squad leader, determine the squad behavior. And from this interaction, squad maneuvers emerge," [8]. Here the emphasis is on emergent team behavior which develops as individual agents share environmental information and select individual courses of action in response to those of their cooperative team members. The obvious problem with this second approach is that its lack of an explicit leader and the self-interested character of the team agents make concerted, strategic team behavior much more difficult to achieve. In response to these two approaches the project was able to identify and pursue a middle approach able to balance the structure of hierarchical organization with the versatility of individual member reasoning.

3 Analysis and Design

Having reviewed past efforts in developing human and team-based AI, the project moved on to its analysis and design stage. Three principal components required design: (1) the human agent architecture; (2) the multiagent system, and; (3) the demonstrative 3D game application.

The project component requiring the most elaborate design was undoubtedly the human agent architecture. The primary challenge in its design was in integrating a variety of stimulus (including emotions, beliefs, and physical perception) with specialized team-based relationship representations and having the resulting system able to make appropriate behavioral decisions for its agent. The architecture itself came to be modeled after Abraham Maslow's famous 'Hierarchy of Needs' (pictured in Figure 2) due to its own integration of emotion and relationship-based reasoning as well as to its ability to balance both reactive and deliberative intelligence. Agent reasoning was thus designed to take place using a tiered, hierarchical arrangement in which the agent would perform one particular type of reasoning at each level. The resulting emotions or observations from each reasoning level would then be collected by the agent and used in selecting the most appropriate action. A given reasoning operation begins at the bottom level (Physiology) and only advances to the upper levels if the assessment results in non-critical values/observations. Arranged in this way, the agent can quickly obviate more complex reasoning when its lower levels determine it is in immediate physical danger.

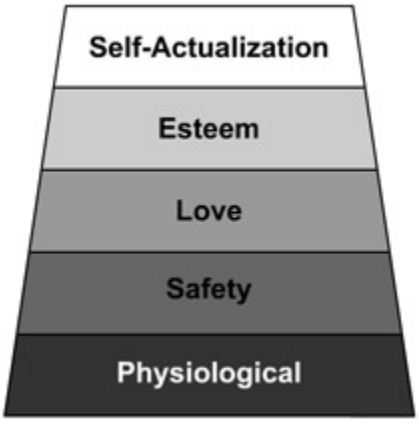


Fig. 2. Abraham Maslow's Hierarchy of Needs [9]

In the following section we describe how this model is used as architecture for our team-oriented agent reasoning model.

4 Agent Reasoning Model

Figure 3 illustrates the design for the agent reasoning model. We can see in the left side the agents' default, low-level reasoning; and in the right side the team-based agent reasoning model [10].

4.1 Team-Based Agent Reasoning

The opportunity for more elaborate agent reasoning is introduced in the designed system once the team leader selects an action for their team to perform. Figure 3 presents a conceptual illustration of this more complex reasoning process, which is the heart of the project work. Here a number of new visual elements are immediately evident. First, the agent reasoning hierarchy is seen as fully-fleshed out in three new levels: (1) a 'Love/Belonging' level; (2) an 'Esteem' level, and; (3) a 'Self-Actualisation' level. At this point it should be evident that Abraham Maslow's Hierarchy of Needs has had a particular influence on the agent's reasoning design. Other noticeable visual elements are the set of emotion parameters positioned on the upper left hand side of the figure. These are the agent's emotions, represented as numeric values normalized over the range of 0 to 1, and linked to a set of emotion functions. Finally, on the right hand side of the figure is a box signifying the design's rule-based system (RBS), which handles its core Planning and action selection functionality.

All of the components and processing pictured in the grey upper square of the figure should be understood as operating within this RBS, but it is conceptually represented as a separate entity to symbolize its role in making the final action selection for the inquiring agent.

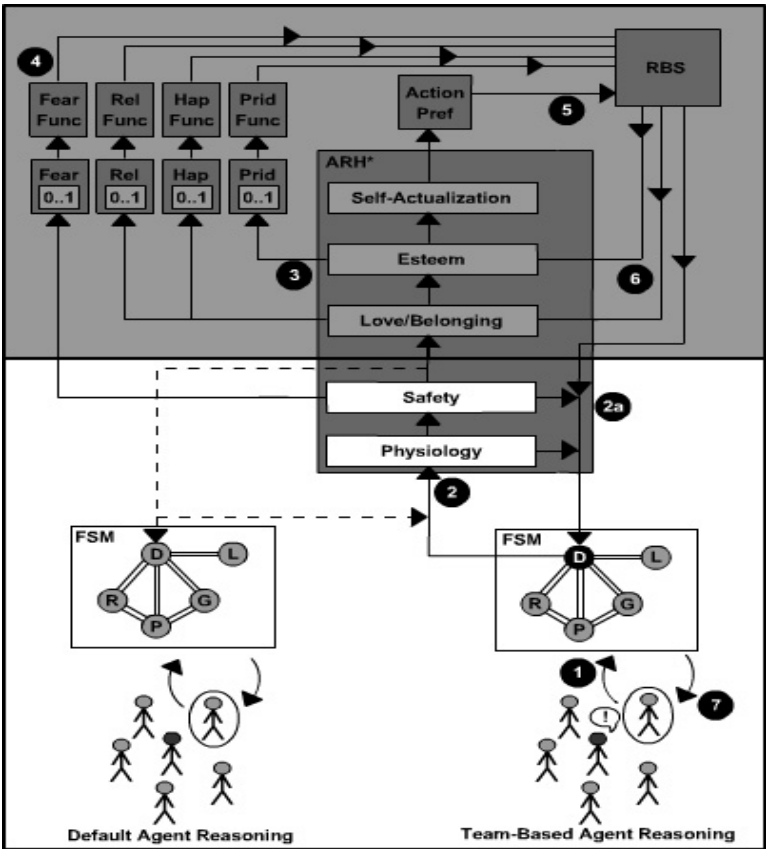


Fig. 3. Overview of Team Based Agent Reasoning

The operation of this more elaborate reasoning design will be demonstrated by walking through the seven-step sequence highlighted on Figure 3. These steps are explained as follows:

(1) The agent looks to the FSM to provide it with an action to perform. The leader of the agent's team has been identified as suggesting an action for the team to perform, and the agent is trying to determine whether it should cooperate or defect and perform its own individual actions. The same reasoning limitations imposed by certain states in the FSM apply, but should the agent be placed into the 'D' state and allowed access to higher level reasoning, the core team-based reasoning of the project can really begin.

(2) The reasoning continues into the agent reasoning hierarchy, where it attempts to scale the various reasoning levels by satisfying their specialized reasoning assessments. The bottom two levels of 'Physiology' and 'Safety' must be surpassed; the project agents are designed to always remain conscious of their physical and environmental conditions in conducting their higher-level reasoning.

(2a) This supplemental step is included to indicate that if either of the two bottom levels of the reasoning hierarchy detect extraneous situations that require immediate response, they can supersede the upper levels and immediately return an action to the agent to perform.

(3) With the agent's physical state secure, reasoning can continue into the upper levels of the hierarchy, which are now managed within the system RBS. The upper levels of the agent reasoning hierarchy are more concerned with processing and updating emotions than recognizing particular environmental conditions. They generally look at previous actions and use recorded stimulus to update the numeric emotion values pictured on the left. The 'Love/Belonging' level is the most important in the hierarchy, as it is where the bulk of the agent's team-based reasoning takes place. It generates two emotions; (1) a relationship strength value ('Rel' in Figure 3) indicating the status of an agent's relationship with their team, and; (2) a happiness value ('Hap' in the figure) that is calibrated relative to the success of the team's collective efforts. The fourth reasoning level, entitled the 'Esteem' level, concerns an agent's self-confidence and generates feelings of pride ('Prid' in Figure 3). It should be noted at this time that the 'Safety' reasoning level generates a fear emotion parameter (provided that reasoning successfully surmounts the level) that can be utilized in higher level reasoning as representative of the danger of the current situation.

(4) Here the normalized emotion values are augmented by a set of agent-specific emotion progression functions. Different agents utilize different functions to model different personality traits. For example, the project provides three functions for the fear emotions interpretation: (1) a 'normal' median function; (2) an enhanced 'coward' function, and; (3) a reduced 'hero' function. These functions allow the basic normalized emotion parameters to be managed in the same manner by the reasoning hierarchy for every agent, while still providing a mechanism for emotional reasoning differentiation. At this step in the sequence the functions are applied and the resulting emotion values are loaded into the RBS in preparation for final action selection reasoning.

(5) The 'Self-Actualisation' level serves mainly as a placeholder in the hierarchy for the final reasoning process. Its main contribution is to load the agent's action preference into the RBS, which is another agent personality detail that will be incorporated into the action selection reasoning. Within the RBS at this step are: (1) the agent's augmented emotional parameters; (2) the agent's action preference, and; (3) the leader's specified action, which is loaded for all of the team agents prior to their team-based reasoning operations. This stimulus is collectively assessed within the RBS at this step, with the final goal of either deciding to cooperate with the leader's suggested action or choosing instead to perform an individual action.

(6) The action selected by the RBS' processing is returned to the agent via the FSM. However, en route the action selection is reviewed by the 'Esteem' and 'Love/Belonging' reasoning levels so that they can better perform their emotional assessments for the next reasoning opportunity. The basic idea is that these levels

look at the action and adjust their team relationship, happiness, and pride emotions depending on whether or not the agent ultimately decided to cooperate or whether the final action was consistent with their action preferences or not.

(7) The agent receives their action instruction from the FSM and carries it out amongst the team agents. The reasoning sequence is complete.

4.2 Finite States Machine

The actual behavior of the human agents was designed to be managed through a low-level finite state machine (FSM).

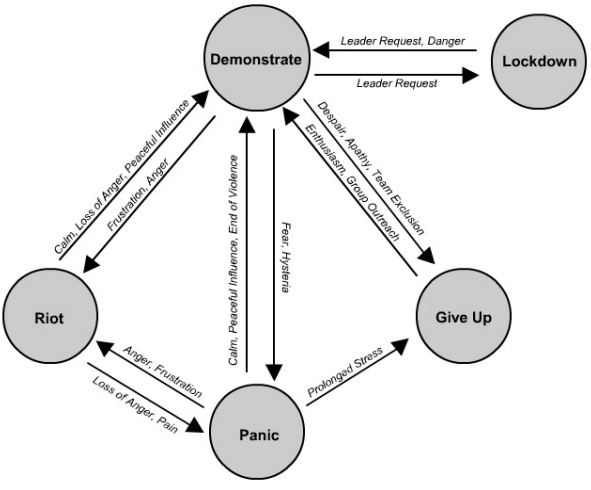


Fig. 4. Agent FSM Design

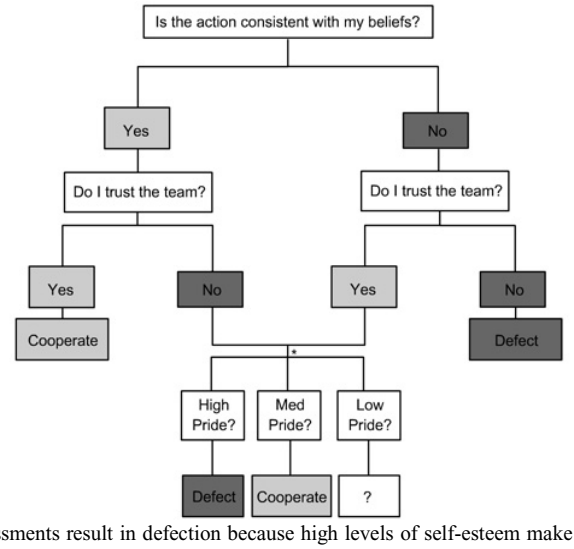
There are a number of reasons for this arrangement. First, the computational complexity of the higher-level reasoning processes was expected to be too significant to carry out for each execution cycle. The FSM provided a means through which a human agent could be placed into a state of action that could continue until its next high-level reasoning opportunity. Second, while the selection of actions by the human agents was intended to be governed through emotional construal, it was thought that this process could be better organized through state representations and transitions.

The FSM for the project's activist agents is pictured in Figure 4.

5 Model Agent Interactions

Agent interaction and relationship modeling is driven primarily through the use of the Iterated Prisoner's Dilemma (IPD) [11].

For the project's modeling of team work situations the IPD was chosen for use as an interesting and powerful means of managing agent interaction and team cooperation. Its adaptation for use in understanding team dynamics is fairly simple. Conditions that influence a team member's decision to cooperate with their team include personal and cultural differences, situational factors, and team dynamics. Those same factors are echoed in the project's adaptation of the IPD. The dilemma is posed to the individual team members when the player-controlled leader proposes an action for the team to carry out together. The individual agents must decide whether they should cooperate with the leader's suggestion or defect and ignore their request, deciding instead to act in a manner of their own choosing. The final factor, team dynamics, is modeled through the use of norms—a concept for individual interaction commonly discussed in social psychology as well as in IPD-based research [11][3].



*High pride assessments result in defection because high levels of self-esteem make the agent take the dominant individual action in either situation. Medium pride assessments result in cooperation because the agent is willing to give the team, or a conflicting action, a chance. The result of low pride assessments depends on which is lower, the agent's happiness or its pride; in low pride situations the agent will try to make the choice that bolsters the lower of the two emotional values.

Fig. 5. Simplified Team AI Diagram

According to the project's design, being a good leader boils down to selecting the appropriate action for the team to perform given their varied personalities and preferences. Such actions will limit defection in these agents because participating in them will positively impact their happiness as well as their pride assessment. Table 1 and Figure 5 are provided to help illustrate the IPD-based team AI design.

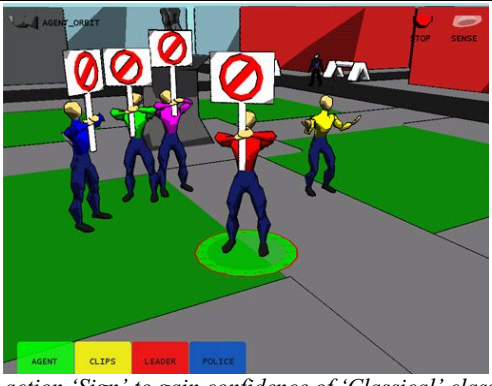
Table 1. Leadership and Team AI Design

Leader Choice	Effect on Agent A		Effect on Agent B	
	Happiness & Love/Belonging	Pride	Happiness & Love/Belonging	Pride
A's Action*	+ + Team majority participation	+ Increase, personal beliefs validated	+ Team majority participation	- Decrease, Personal beliefs betrayed
	- Team minority participation		- - Team minority participation	
B's Action*	+ Team majority participation	- Decrease, personal beliefs betrayed	+ + Team majority participation	+ Increase, personal beliefs validated
	-- Team minority participation		- - Team minority participation	
Other Action (careless)	+ Team majority participation	- Decrease, personal beliefs betrayed	+ Team majority participation	- Decrease, personal beliefs betrayed
	- - Team minority participation		- - Team minority participation	

6 Demonstrate! Game Application

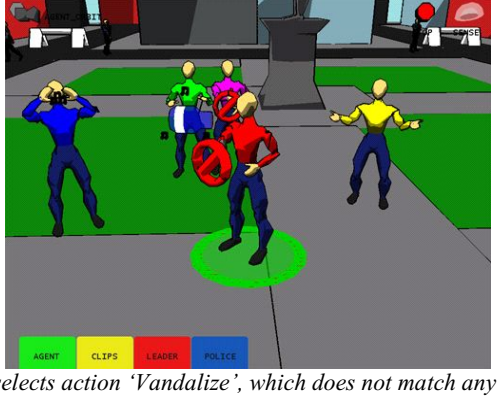
The design of the demonstrative 3D game application—seen as the key to illustrating the successful operation of the complex human agent architecture to interested parties—was developed alongside the artificial intelligence it was intended to support (a screenshot of the application developed as test-bed can be seen in Fig 6).

Ironically entitled 'Demonstrate!' the game was intended from very early stages in the project's development to involve a political street demonstration setting. The political demonstration setting was seen as providing the potential for a variety of agent personality types and motivations, a vast repertoire of different actions and emotional expression, and an interesting new context for developing teamwork situations and reasoning.



Leader selects action 'Sign' to gain confidence of 'Classical' class team member

The police agents scatter the activist team, using aggressive physical force and causing the agents to respond with evasive actions.



Leader repeatedly selects action 'Vandalize', which does not match any of the team agents' classes and the destructive relationship effects become evident in the reluctance of the majority of the agents to cooperate

Fig. 6. "Demonstrate!" Application screenshots

More detailed information about the design and implementation of the model, along with an extensive set of scenarios and evaluation can be found in [10], showing the relative success of the model and the agents interaction.

7 Conclusions

"The Development of a Cooperative Multiagent System to Facilitate Leadership Roles in Computer Entertainment" project [10] set out to investigate agent-based artificial intelligence techniques to simulate realistic team dynamics for use in computer gaming.

The success of the developed AI and its expression within the demonstrative 3D game application are indicative of the many strengths of the work. The constructed reasoning design was successful in achieving a good level of team-based agent reasoning and behavior, as demonstrated in the sections on its evaluation [10]. Leadership and team dynamics remain a complex challenge for artificial intelligence developed within the academic AI community or the games industry. Some initial steps have been made by the project work, but significant challenges lie ahead.

Beside the potential application of the model and architecture to a computer entertainment environment, the model is generic and can be used as well for "serious" application which involves distributed emerging behavior, scenarios based simulation, complex agent-based modeling including emotional, reactive and deliberative reasoning.

The authors can conceive of a huge number of areas in which the project work can be improved. The developer sincerely hopes that it may prove to be of some merit in influencing new research directions or game designs that recognize the potential that resides in teamwork and leadership as resources for driving future innovation in artificial intelligence.

References

1. Gibson, Clark, and O'Brien, John. "The Basics of Team AI," Game Developers Conference Proceedings, 2001. (17 Nov 2003) www.gdconf.com/archives/2001/o'brien.ppt.
2. Laird, John E and Pottinger, David C. "Game AI: The State of the Industry, Part Two" Gamasutra.com. 8 Nov 2000. http://www.gamasutra.com/features/20001108/laird_03.htm .
3. Brehm, Sharon S., Kassin, Saul M., and Fein, Steven, (2002), Social Psychology, Houghton Mifflin Company, Boston,, USA.
4. Musse, S.R., and Thalmann, D. "A Model of Human Crowd Behavior Group Inter-Relationship and Collision Detection Analysis", Citeseer. (1997). (19 Jun 2004) http://citeseer.ist.psu.edu/musse97model.html - http://citeseer.ist.psu.edu/bates94architecture.html
5. Bates, J., Loyall, A.B., and Reilly, W.S. "An Architecture for Action, Emotion and Social Behavior," Citeseer. (1994) Brehm, Sharon S., Kassin, Saul M., and Fein, Steven. Social Psychology. Boston, MA, USA: Houghton Mifflin Company, 2002.
6. Silverman, B.G., et al. "Human Behavior Models for Game-Theoretic Agents: Case of Crowd Tipping", Citeseer. (2002) http://citeseer.nj.nec.com/silverman02human.html

7. Reynolds, J. “Tactical Team AI Using A Command Hierarchy”, AI Game Programming Wisdom, Rubin, S. Hingham, MA, USA: Charles River Media, Inc., Hingham, MA, 2002.
8. Van Der Sterren, W. “Squad Tactics: Team AI and Emergent Maneuvers”, AI Game Programming Wisdom, Rubin, S. Hingham, MA, USA: Charles River Media, Inc., 2002.
9. Maslow, Abraham H. Motivation and Personality. New York, NY, USA: Harper & Row Publishers, Inc., 1954.
10. Nick Baker and Abdennour El Rhalibi (2004), The Development of a Cooperative Multiagent System to Facilitate Leadership Roles in Computer Entertainment, ACM GDTW 2004 International Conference, Liverpool John Moores University. November 2004.
11. Axelrod, Robert (1957), The Evolution of Cooperation, Basic Books, Inc., Publishers, NY, USA.
12. Reilly, W. Scott, and Bates, Joseph, (1992), “Building Emotional Agents”, Technical Report CMU-CS-92-143, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA.

Defining Agents Via Strategies: Towards a View of MAS as Games

D.R. Vasconcelos[1], E.H. Haeusler[1], and Mario R.F. Benevides[2]

[1] Pontifícia Universidade Católica do Rio de Janeiro, Brazil
davirv,hermann@inf.puc-rio.br
[2] Programa de Sistemas, COPPE/UFRJ, Brazil
mario@cos.ufrj.br

Abstract. In this article, we intend to characterize, at least on BDI (Belief-Desire-Intention) basis, a class of games G for which there is a MAS and vice-versa. As a consequence, criteria of rationality for agents can be directly applied to players and vice-versa. Game analysis formal tools can be applied to MAS as well. The main results of this article are the following two lemmata.
Lemma I: Every MAS belonging to G is, essentially, a Game.
Lemma II: Every Game can be implemented as a MAS and Equilibria are Optimal Desires Satisfaction.

1 Introduction

Multi-Agent Systems (MASs) have been widely used as an adequate abstraction for the design and implementation of a large sort of applications. A system of agents whenever appropriately arranged is called a community of agents. This arrangement concerns the integration of *individual beliefs* towards *common beliefs*, integration of agent's *individual plans* into *common plans* and finally the integration of agent's *individual actions* performing *common actions*. This integration follows and is induced (in a very precise mathematical meaning) by the way the community is organized, that is, by the way each kind of agent communicates each other and the way that they can communicate their beliefs and interact among themselves. This establishes a kind of architecture for a MAS. This article regards to the view that a MAS is strongly based on the integration of the belief-intention-planning of the individual agents in order to build a whole community of agents. Specific aspects of action and ontological (structured) concerns of MASs are out of scope of this report.

Software Architecture is a well-established and studied subject whenever concerned to components. Let us discuss briefly some points that raise when comparing MAS and Component-Based architectures. Differently from Components, Agents have a belief model, and a set of intentions, that rules plans, which are supposed to perform actions in order to satisfy some of their intentions. The interdependence among intentions, the set of beliefs and the generated plans

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 299–311, 2006.

become the behavior of an agent more sophisticated than a sole component behavior. Even for a reconfigurable component architecture, the behavior of their components is more rigid than the agent case. In general a dynamic reconfiguration is obtained by changing the channels of communication among the components. This can be done by changing the connectors in a predicted way (a connector has a finite amount of ports). On the other hand, despite the amount of possible paths of communications among agents being finite as well, the way of a MAS changes the channels of communications is strongly related to the belief-intention-planning feature of the agent, and this is less predictable than the component's case, since the evolving of an agent's belief is a feature more related to logic than to computation (in the sense of behavior). The literature call BDI (Belief-Desire-Intention, see [9]) model to this logical approach that integrates belief, intention and planning in order to model the main concerns of an individual agent.

We will take the BDI model as our starting point for an individual agent definition. Regarding the MAS architecture discussion, we take as an essential aspect to be dealt with the integration of the very logical aspect of each BDI agent model. Thus, we take a MAS, in the context of belief-intention aspects, as the harmonious integration of each individual agent's model in the whole system[1]. As one can expect, an (individual) agent of a MAS does not have to believe or even know the global intention of it. The MAS behavior should be essentially driven by the rational aspect of each of its components. This very feature regards (rational) players in a game, from the particular point of view of Game Theory.

Game Theory is a branch of Mathematics, extensively used by economists and social scientists, well-suited to define and study rational behavior in the context of cooperative and non-cooperative environments. Equilibria notions (after Nash) play a central role in solution concepts regarding a great variety of Games assuming the players are rational. Game Theory is mainly taken into account when the scientists want to predict and study global phenomena from individual known behavior. Typical examples of this kind of research come from phenomena emerging from Markets, Auctions and Elections. One can observe a strong similarity between the rationales coming from Game Theory and Multi-Agent Systems modeling. Game Theory provides some general Game definitions, some of them equivalent in a certain sense. It also provides solution concepts to each kind of Game, pointing out relationships among the different solution concepts, as for example the "Folk Theorems" relating equilibria points in a game to the repeated game version. Algorithms for searching and formally identifying solution concepts in a (generic) game are worth of study for theoretical and practical notions. It is known that the decision problems associated to most of the Game

[1] An interesting and natural consequence of this view of a MAS is that it has also its own BDI model, representing common beliefs, intentions and performing a common plan to fulfill its goals, taking a coalition of agents as a new kind of agent. However this broader view of MAS, as a single agent, seems to be of no help in practical design of agent-based systems.

Theory solution concepts belong to higher complexity classes (starting with the NP class, of course). Apart from that, it seems to be no better alternative for the social sciences, mainly when analytical[2] tools do not apply.

The previous discussion raises up the point of having Game Theory as a worth tool to be applied in the analysis of MAS validation and designing. The existence of formal tools (programs), some of them listed in a subsequent subsection, that help in this validation task turns out feasible the formal validation of MAS. On the other hand, MAS themselves can be applied to model and formally study, by means of simulation, the kind of phenomena typically subjected to Game Theory analysis. In [1] it is shown a quite good comparison between Game Theory and MAS formal analysis of the market of electricity. The conclusion one could draw from these raised points is that Games and MAS are in a 1-1 relationship. This, not yet drawn, conclusion is the main guideline and motivation of this article. We intend to characterize, at least on a BDI basis, a class of Games for which there is an equivalent MAS and vice-versa. The notion of equivalence of MAS and Games will be taken as logical, since BDI has a logical counter-part played by *Lora*'s[3] theories.

1.1 More Motivations to Our Goal

By taking the main defining aspects of the BDI model as specific strategies in an extensive game (see [6]) and the behavior of an agent as a player's (possible) sequence of steps in this game, we intend to show that in some cases, a MAS can be viewed as a kind of coalition game with the coalition among the players representing the integration of each player individual strategy into the whole strategy of the coalition game. From that point on, one can proceed a formal analysis (verification, validation of properties as well as theorem proving) of a MAS using formal tools designed for game solution concepts analysis.

In [3] a logical view of game-analysis is shown. A logic, GAL, for Game Analysis Logic, based on first-order CTL (computational tree logic) is defined and used to specify solution concepts as well as ordinary properties on specific games.

Although historically Game Theory has been considered more suitable to perform quantitative analysis than qualitative ones, there has been a lot of approaches that emphasizes Game Analysis on a qualitative basis, by using an adequate modal-logic in order to express properties of the games as well as their solution concepts (existence of Nash-equilibria, core, etc). One of the most representative of these logics is ATL (Alternating-time Temporal Logic), and, we point out [7,8] as excellent references on this approach. However, since ATL is a propositional logic language and BDI has been firstly developed on a first-order setting using *LORA* (see [9]), we choose to develop our approach using GAL itself, since it has first-order features and is a logic language which can express strategies in a suitable way. In [4] it is shown how GAL is able to specify Strategic Games and Coalition Games with Transferable Payoffs, as well as their main

[2] Based on the differential/integral equations mathematical paradigm.

[3] *LORA* is the logical language designed to specify agents regarding to the BDI model.

solution concepts, namely, Nash Equilibrium and Core. Besides that, we have a prototype of model-checker for GAL that has been developed according to the main intentions of the approach advocated here.

This article argues in favor of taking agents as players in a game and that the definition of an agent is made by the definition of his strategy in the game. The main results of this article are to characterize a class G of MAS as Games such that the following two lemmata hold.

2 The Background Theories: BDI and Game Theory

It is assumed a basic reader's familiarity with first-order and modal logics, including Kripke models and Bisimulation.

In order to have a theoretical basis to use Game-Analysis tools to proceed the formal analysis of agents and MAS, we need to argue that from the very starting of the design of an individual agent, we can take it as a player in a specific game. We base our argumentation on the BDI model for agents. The BDI model is based on three logical modalities, namely, the Belief modality, the Desire modality, the Intention modality, forming a Multi-modal logic, the first modality is a *KD45* modality (see [5]), the other two are *KD* modalities. GAL is based on CTL and CTL has the $\exists X$ ("in some next state") modality as a KD modality and $\exists U$ ("until") modality behaving as a KD4L fixed-point modality over $\exists X$. Thus, in principle, the expressiveness power of both logics are comparable, not forgetting that both are first-order logics. However, the Barcan principle, as well its inverse, that are required in GAL, seems to be liberalized in LORA (the logical counter-part of the BDI model). The temporal aspects are different, since LORA is based on CTL^*, while GAL is based on CTL. However, since GAL has only Games as models, this makes inessential the small differences discussed above. The first lemma is a way of expressing, by means of strategies, the Desire-Intention aspect of BDI, while ensuring that the states of the defined game provide enough Belief for the player.

2.1 The BDI Model

In this subsection we state the basic terminology, definitions and notations of the BDI model for (individual) agents. With the sake of a succinct presentation we use some concepts of the possible worlds semantics of the logical language *LORA*, presented in [9]

This article does not intend to provide new definitions for BDI-agents. Thus, the mathematical terminology used in the sequel only has the purpose of providing a shorter text. The following definition is an extensional (model-theoretic) counter-part of the well-known BDI intentional definition such the one found in [9]. However, the reader must take into account that there is an intentional meaning for the agent. This intentional meaning has the role of establishing a relevant distinction between Beliefs, Desires and Intentions, namely:

- Beliefs consist of information available to the agent.
- Desires are "things" that the agent would like to make true. The only restriction on desires is the consistency. Agents cannot have inconsistent desires.
- Intentions are desires that the agent has chosen and committed to.

Definition 1. *A temporal structure* $< T, <_T >$ *is a set* T *and a partially, discrete and serial order on it*[4]. *A temporal sub-structure* $< T\prime, <_{T\prime} >$ *of* $< T, <_T >$ *is such that* $T\prime \subseteq T$ *and* $<_{T\prime} \subseteq <_T$.

Sometimes we use the carrier set T when referring to a temporal structure $< T, <_T >$.

We remind the reader that a first-order language is a set of functional and predicate symbols of any finite arity[5]. Since we deal with computational agents, the following definition of MAS restricts the first-order language to a finite set of symbols, both predicates and functionals. In that case we name it as a finite first-order language.

Definition 2. *Given a finite first-order language* $\mathcal{L}$ *we call* $\mathcal{L}$*-state a pair* $< S_{\mathcal{L}}, \sigma >$ *where* $S_{\mathcal{L}}$ *is first-order structure*[6] *for* $\mathcal{L}$ *and* $\sigma : Vars \rightarrow Dom(S_{\mathcal{L}})$ *is an assignment of values for variables.*

Definition 3. *Let* $< T, <_T >$ *be a temporal structure. A click over* T *is a pair* $(t_1, t_2) \in T \times T$, *with the property that there is no* $t \in T$ *such that* $t_1 <_T t <_T t_2$ *and* $t_1 \neq t$ *and* $t_2 \neq t$. $Clicks(T)$ *is the set of all clicks over* T

Definition 4. *A MAS over a first-order language* $\mathcal{L}$, *with a set of agents* A, *and (finite) set of actions* X, *is a structure of the form* $< T, W, (B_a)_{a \in A}, (I_a)_{a \in A}, (D_a)_{a \in A}, (S_w)_{w \in W}, Ac_X, Ag_X >$, *such that: 1-* W *is a set of temporal substructures of* T; *2-* S_w *is a set of* $\mathcal{L}$*-states indexed by the time structure* T_w; *3-* $B_a, D_a, I_a \subseteq W \times T \times W$ *are, respectively, the Believe, Desire and Intention Kripke relationships between the states, for the agent* a; *4-* $Ac_X : Clicks(T) \rightarrow X$ *associates an action with each time step in* T; *5-* $Ag_X : X \rightarrow A$ *associates an agent with each action in* X. *The following restrictions on the structure must hold:*

- *For each agent* a, *if* $< w, t, w' > \in B_a$ *then* $t \in T_w \cap T_{w'}$. *Analogously for* D_a *and* I_a *respectively.*
- *Define* $B_a^{t,w} = \{w' / < w, t, w' > \in B_a\}$, *and* $< w, w' > \in B_{a,t}$, *iff,* $w' \in B_a^{t,w}$. *For each* a *and* t, $B_{a,t}$ *must be a serial, transitive and euclidean*[7]. *Analogous we define* $D_{a,t}$ *and* $I_{a,t}$ *and these relations must be serial for each* a *and* t.

In the above definition a MAS is identified with its semantics. A state S_t represents the description of the MAS at time t. A world $w \in W$ represents the beliefs on the evolving, in time, of state descriptions.

[4] A relation $<_T$ is serial, iff, $\forall t_1 \exists t_2 (t_1 \neq t_2 \wedge t_1 <_T t_2)$.

[5] A functional symbol with arity 0 (zero) is called a constant.

[6] Space limits prevent us to write down the usual definitions regarded to first-order logic semantics.

[7] R is euclidean, iff, if $< w, w' > \in R$ and $< w, w > \in R$, then $< w', w' > \in R$.

For the sake of terminology and notation, we denote as $sub(t, w)$ the subtree of T_w determined by the node t, whenever $t \in T_w$.

It is worthwhile mentioning that the above extensional definition is the semantics of an intentional definition of the MAS, usually defined by means of a programming of each of its agents in a loop-control as, for example, the one shown in algorithm 1. It is well-known that this control-loop can be more detailed, such that it includes assessment and reconsideration of Intentions, soundness of Planning with Desires, and more. The algorithm 2 shows a refined version of the previous one. Both versions are from [9] and can be seen as particular cases of the semantic Planning function defined below. This article takes as free the self-explanatory content of these algorithms. The reader must observe that, according to the BDI definition taken in this article, there may be simultaneous actions in the MAS.

Algorithm 1. Agent Control Loop

$B := B_0$;
$I := I_0$;
while true **do**
  get next percept ρ;
  $B := brf(B, \rho)$;
  $D := options(B, I)$;
  $I := filter(B, D, I)$;
  $\pi := plan(B, I)$;
  $execute(\pi)$;
end while

The reader is invited to see the agents intentionally defined above as denotations, according to the MAS definition 4, of either version of the shown agent control-loop algorithms. The intuitive meaning of these control-loops, regarded to the MAS extensional definition can be understood as following:

- The beliefs of an agent are almost dependent on the evolving of the time, since this is the only effect that can cause new percepts. It is worth noting that the evolving time can be caused by an action of (possibly) other agent;
- The Desires depend on the (new) beliefs and the (old) intentions. This means that from a (hypothetical) belief position the agent may go further when considering intentions (next item);
- The (new) intentions that can be taken by the agent include those can be considered by an action raised hypothetically by the agent. This is very similar to a look-ahead "attitude";
- Re-considerations may come and the consequent soundness analysis is among the tasks towards efficiency.

In fact an agent planning can be thought as a procedure that evaluates the set of Beliefs, Intentions and Desires of an agent and produces a structured action

Algorithm 2. Refined Agent Control Loop

```
B := B_0;
I := I_0;
while true do
  get next percept ρ;
  B := brf(B, ρ);
  D := options(B, I);
  I := filter(B, D, I);
  π := plan(B, I);
  while not (empty(π) or succeeded(I,B) or impossible(I,B)) do
    α := hd(π);
    execute(α);
    π := tail(π);
    get next percept ρ;
    B := brf(B, ρ);
    if reconsider(I,B) then
      D := options(B, I);
      I := filter(B, D, I);
    end if
    if not sound(π,I,B) then
      π := plan(B, I)
    end if
  end while
end while
```

(a term in an algebra action). This structured action will afterwards executed subterm by subterm, sometimes yielding concurrent actions applications. However, each time either Beliefs, Intentions or Desires sets are update, the currently stored plan is revised to maintain soundness. From an abstract setting, the planning is a mapping from Beliefs, Intentions and Desires into evolving $\mathcal{L}$-states, so to say, a temporal substructure of T. Of course this is a simplified version of planning, since we are taking a planning application as an atomic procedure. We briefly discuss at the conclusion how to consider the more sophisticated setting at the theoretical level, without going too abstract. The *Plan* function, defined below is the semantic counterpart of the just discussed concept of planning. Considering the planning as a relation instead of a function, in order to get into non-deterministic plan generation is of courser formally feasible in our setting, but this would make the notation and definitions heavier. More than that, the *Plan* semantics function, for reasons of consistency, is a partial function.

In the following definition we consider $Bel_a = Image(B_a) \cup Range(B_a)$ and analogously for Des_a and Int_a.

Definition 5 (Planning Semantic Function). *Let* $M =< T, W, (B_a)_{a\in A}, (I_a)_{a\in A}, (D_a)_{a\in A}, (S_w)_{w\in W}, Ac_X, Ag_X >$ *be a MAS. A planning function for the agent* $a \in A$ *is a partial function* $Plan_a : Des_a \times Int_a \times Bel_a \rightarrow Int_a \cup Bel_a$,

such that, for all $t \in T$, *if* $< w, t, w_1 > \in B_a$, $< w, t, w_2 > \in I_a$, $< w, t, w_3 > \in D_a$ *then* $Plan_a(w_1, w_2, w_3)$ *is defined.*

The following definition does not apply to all kinds of BDI MAS, it is restricted to those which do not allow simultaneous actions applications belonging to different agents. The reason for this restriction is organizational. In this article we consider this kind of MAS with the sake of a simpler presentation of the relation between Games and MAS. At the conclusion we discuss the extension of the obtained results to a broader class of MAS. We call a MAS under the previous restriction as one-agent-turn MAS or OAT MAS.

Definition 6. *Let* $M =< T, W, (B_a)_{a\in A}, (I_a)_{a\in A}, (D_a)_{a\in A}, (S_w)_{w\in W}, Ac_X, Ag_X >$ *be a OAT MAS.* $Plan_a$, *for each* $a \in A$ *the Planning (partial) functions of each agent planning. The situation tree of* M, *regarded to the family of plans* $Plan_a (a \in A)$ *is defined as the labelled rooted tree* $Sit_M(Plan_a)$ *that comes from the application of the following steps:*

1. *The root of* Sit_M *is the same root of* T *and* $Sit_M(Plan_a)$ *is initially taken as* T*;*
2. *The labels of* Sit_M *are determined by* Ac_X*;*
3. *If, for an agent* a, $< w, t, w_1 > \in B_a$, $< w, t, w_2 > \in I_a$, $< w, t, w_3 > \in D_a$ *and* $w_p = Plan_a(w_1, w_2, w_3)$, *then* t *has* $sub(t, w_p)$ *as a subtree of it in* $Sit_M(Plan_i)$*;*
4. *If there is a time* $t \in Sit_M$, *such that, there are two or more Clicks labelled with the same action* $v \in X$ *leading into times* $t_1, \ldots, t_k$ *and* $\{S_{t_1}, \ldots, S_{t_k}\}$ *is not a chain*[8], *then replace all of these subtrees by an infinite path of clicks labelled by* v[9]*;*
5. *Each non-terminal node* u *of* $Sit_M(Plan_a)$ *is labelled with* $a \in A$, *such that, for all* $t \in Clicks(T)$ *that are outgoing branches of* u, $Ag_X(Ac_X(t)) = a$ *holds.*

One can think of $Sit_M(Plan_i)$ as describing all alternatives of evolution of the behavior of M, regarding the plans $Plan_a(a \in A)$. In the sequel (section 3) we obtain a game definition from $Sit_M(Plan_i)$.

2.2 Game Theory Basics

Space limitations prevent us to mention the rationale of the definitions related to Game Theory. In the sequel, we write down the definitions and theorems used in this article. Mathematical definitions used in this article to prove the lemmata of the next sections can be found in [6] and [2].

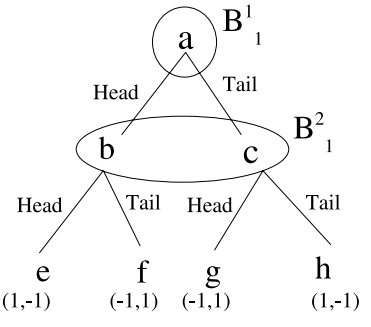
[8] A set $\{A_1, \ldots, A_k\}$ of $\mathcal{L}$-structures is a chain, iff, there is a permutation $p : [1, k] \to [1, k]$, such that $A_{p_1} \leq A_{p_2} \leq \ldots \leq A_{p_k}$, where $\leq$ is the substructure relation.

[9] The fact that the set of $\mathcal{L} - Structures$ S_{t_i} is not a chain means that if the MAS evolves by taking the v action then there will be no $\mathcal{L}$-structure ensuring the semantics of the system.

Definition 7 (Kuhn1953). *A n-person game in an extensive form, extensive game for short, Γ is a structure consisting of the following data:*

- *A finite set $N = \{1, 2, \ldots, n\}$ of players.*
- *A (rooted) tree γ, the game tree.*
- *A partition of the set of non-leaf nodes of γ into $n+1$ subsets $P_0, \ldots, P_n$. The nodes belonging to P_0 are called chance nodes, since they have probabilities attached to their outgoing branches. For the other subsets, P_i is the set of (possible) moves of player i in γ.*
- *For each node p in P_0 a probability distribution over the outgoing branches of p.*
- *A finite set of actions α. Each outgoing branch, from a non-leaf node, is labelled with an unique element of α, except the chance branch which is labelled with a real number (a probability).*
- *For each $i \in N$ a partition of P_i into $n(i)$ information sets, $B^i_1, \ldots, B^i_{n(i)}$, such that for every $j = 1, n(i)$:*
  - *All nodes in B^i_j have the same number of outgoing branches, with a 1-1 correspondence between the sets of respective outgoing nodes of different nodes in B^i_j.*
  - *Every path in the tree from the root to a leaf-node visit each B^i_j at most once.*
- *For each leaf-node t, an n-dimensional vector $h(t) =< h_1(t), \ldots, h_n(t) >$ of payoffs.*
- *The structure just being described is common knowledge among the players.*

The last item characterizes the class of games of perfect information. In a game of imperfect information this clause is not included. Figure (a) shows the extensive form of the Matching Pennies game.



(a) - Matching Pennis game

3 Agents as Players

We want to move toward a Game-Theoretical foundation for *MAS*, mainly in the context of computational agents. Thus, in this article we focus only on this

kind of MAS. Natural agents and Games running on concrete (Real) entities are left outside the scope of this work.

In what follows, agents will always be taken as computational agents, as well MAS refers to a system of agents in the sense of 4. Consider an agent A in a MAS $\mathcal{M}$. The main analogy between agents and players in a game is depicted by the following association.

Agent Concept	Game Concept
Beliefs	State-Description
Intention	Strategy
Desires	Maximization of Payoffs [a]

[a] The assignment of payoffs will be based on the extensional semantics of the MAS itself.

We will see that the payoff assignment will be the key of this relationship, on the other hand it can be seen as natural from the extensional point-of-view. A MAS, in general, yields a repeated game. The following discussion would be better understood if the reader could consider the portion of the MAS associated to the stage game. Anyway, the analogy between MAS and Games can be observed in the following table.

Agency Theory	Game Theory
Agents Groups	Players
Common Beliefs	Game Tree
Agent's Intentions	Player's Strategies
Desires Satisfaction [a]	Existence of Equilibria

[a] Here we refer to the satisfaction of desires of a set of agents.

Before we show how to assign a game to MAS M, we have to take the care of removing infinitely long paths in $Sit_M(Plan_i)$ that are meaningful in the game-theoretical interpretation. These paths consist of $\mathcal{L}$-states that are bissimilar. As it happens with infinitely long paths, two cases are possible:

- There is no repeated elementarily equivalent $\mathcal{L}$-state in the path.
- There is at least one repeated elementarily equivalent $\mathcal{L}$-state in the path.

The first case will never result in a finite path if bissimilar states are identified. The other case, however, may lead to a finite path, just in case that the bissimilar state that repeats infinitely often is the one that causes the infinite feature of the path. This seems to be strongly related to the well-known "Stage-Game" in an infinitely repeated game. This discussion motivates the following definition. This will be used in the game-tree definition of the game associated to a MAS as states lemma 1.

Definition 8. *Consider a MAS M over a first-order language $\mathcal{L}$ and $Sit_M(Plan_i)$ as stated by definition 6. Given two elementarily equivalent $\mathcal{L}$-states $s1$ and $s2$, we say that $s1 \equiv s2(mod(Sit_M(Plan_i)))$, iff, $s1$ and $s2$ are bissimilar.*

It is a routine task to check that the definition above is an equivalence relation. Thus we define $SIT_M(Plan_i)$ as the quotient of the transition system $Sit_M(Plan_i)$ regarded to $\equiv$, with the addition of the following: Any transition that, after the quotient, has the same target and source will be removed. The reader can observe that the later makes $SIT_M(Plan_i)$ a tree.

The following definition establishes in a more detailed way the relationship depicted by the above tables.

Definition 9. *Let M be a MAS over a first-order language $\mathcal{L}$ with (finite) set of agents A, (finite) set of actions X and family of plans $Plan_a(a \in A)$, then there is an extensive Game G_M, such that, each agent A of M corresponds to a player p_A, in a way that :*

- *The game tree of G_M is the situation tree $SIT_M(Plan_i)$.*
- *The partition of the non-terminal nodes of $SIT_M(Plan_i)$ is $P_0 = \emptyset$ (there is no chance node at all). P_a is the set of all nodes labelled with a.*
- *The partition of P_i, for each i, is $B_i^1 = P_i$.*
- *The set of strategies are determined by the branches in a free, in the mathematical sense, way.*
- *The payoff of a terminal node, for each agent a, is the number of desires of a that are satisfied by the $\mathcal{L}$-state associated to this terminal node. Thus the payoff vector has d_a as the component a, where d_a is the number of satisfied desires.*

Lemma 1. *In the conditions of the above definition, each subgame equilibria of G_M expresses a $\mathcal{L}$-state where every possible desire for each agent is satisfied.*

Proof-Sketch. A subgame equilibria in G_M is a node in $SIT_M(Plan_i)$ that represents a position in the behavior of the M, such that, any further evolution (in time) of M puts the system in a state that cannot evolve to a state with more desires satisfied, not considering the other agent moves. On the other hand, any state in M that has an optimal number of desires satisfied for each agent a, independent of the other agents evolution in time, by the construction of $SIT_M(Plan_a)$ and G_M is a subgame equilibria in the later. Thus, the subgame perfect equilibria correspond to states where the agents have optimal satisfaction regarding to desires, that is, are states which have optimal social satisfaction.□

For the following lemma does not matter whether the game is or not is in an extensive form. However, in both lemmata the games are of perfect information. This hypothesis should be proved to be non-restrictive, but this result has not been worked yet.

4 Players as Agents

On the other hand, the following lemma is simpler, since any player in a Game (extensive form or not) can be modeled by an Agent in a MAS representing the game itself. As a matter of fact, the definition of the MAS follows the relationship stated by the tables of the previous section. The fact that some games are not computationally representable does not bother the association of MASs to games. Only computational games can be implemented as MASs. The image of a computational game is a (computational) MAS, hence, may belong to the class G of OAT MASs. Indeed, any computational game, with non-simultaneous moves, is mapped into MAS that belongs to G as the following lemma holds.

Lemma 2. *Let g be a computational game, with non-simultaneous moves, then there is a MAS $M_g \in G$ that implements g in a way that the desires of M_g are the maximization of the payoffs of the game, the intentions are the strategies and each player corresponds to an agent. Equilibria (subgame perfect equilibria) of the game are states of optimal social satisfaction*[10]

The proof sketch for lemma 2 is as follows

- Each agent corresponds to a player.
- The desire of the player is to maximize the payoff.
- Beliefs are state descriptions of the extensive game.
- The strategies of each player determine the Planning of the each agent.
- States of maximal social of M (if any) are subgame perfect equilibria of the game.

5 Conclusion

This article shows that for Games and OAT MAS are equivalent models of behavior and rationality. As a consequence, criteria of rationality for agents can be directly applied to players and vice-versa. Game analysis formal tools can be applied to MAS as well.

It seems to be a routine task the extension of the results mentioned here to general MAS (non-OAT) and games that allow the simultaneous actions of the players, as the models discussed in [6] (pp. 102–13). Other way of dealing with MAS with the occurrence of simultaneous actions, from different agents, is the construction of the situation semantics ($Sit_M(Plan)$) taking into account sets of agents as nodes. In this way the situation semantics should be associated with a Coalition game (see [6]), a cooperative model of games. The solution concept that must be used in this case is that of *core*. Fortunately, *cores* seem to be associated to the equilibria states of an extensive Coalition game associated to $SIT_M(Plan)$, when the payoffs are taken according to the coalitions that appear

[10] A state of optimal social satisfaction is a state where each agent has the most possible, whatever action of other players, satisfaction.

in the situation semantics. In fact, when agents cooperate among them, it is natural that common desires, intentions and beliefs emerge. A consequence of this is that the game would modify the way the payoffs are assigned. Coalition games with transferable payoff seem to be the adequate model for this case. One of the future direction of our research is the investigation of the following **conjecture**

> Every MAS can be viewed in the form of a Coalition Game with Transferable Payoff

such that the core is the semantics of the MAS.

References

1. T. Krause, G. Andersson, D. Ernst, E.V. Beck, R. Cherkaoui and A. Germond: A comparison of Nash equilibria analysis and agent-based modelling for power markets. In Proceedings of the 15th Power System Computation Conference (PSCC 2005). Lige, Belgium, 22-26 August 2005
2. Aumann, R. and Hart, S.: Handbook of Game Theory. Vol. 1. North-Holland, 1992.
3. Vasconcelos, D. R. and Haeusler, E.H.: A Logic view of Playing Games. In *Frontiers in Artificial Intelligence and Applications*, volume 101, pages 67-80, IOS Press, 2003.
4. Vasconcelos, D. R. and Haeusler, E.H. and Poggi, M. and Benevides, M.F.: Reasoning about games via temporal logic: A model checking approach. 13 p. PUC-RioInf.MCC47/04. ISSN 0103-9741.
5. Goldblatt, R.: Logics of Time and Computation. CSLI lecture notes 7. Stanford, 1987.
6. Osbourne, M.J. and Rubinstein, A.: A Course in Game Theory, MIT Press, 1994.
7. Parsons, S. and Wooldridge, M: Game Theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243-254,2002.
8. Kacprzak, M. and Penczek, W.: Unbounded model checking for alternating-time temporal logic. In *Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS04*, pages 646-653, 2004.
9. Wooldridge, M.: Reasoning about Rational Agents. MIT press, 2000.

Secure Mobile Agent Deployment and Communication Towards Autonomous Semantic Grid

U. Topaloglu[1], C. Bayrak[2], and N. Kanaskar[1]

[1] Applied Science Department
[2] Computer Science Department,
University of Arkansas at Little Rock
Little Rock,AR 72204
{umtopaloglu,cxbayrak,nvkanaskar}@ualr.edu

Abstract. As the information technology rapidly expands (regardless of the technology area) collaboration, information sharing and distributed computing become inevitable phenomena (i.e. e-commerce, e-health, e-science etc.) [1]. Since the complexity is a driving force behind the enhancements, new network-centric infrastructures and solutions are proposed to address the problem. Grid is an example solution that handles the situation in a cost-effective and efficient manner. There are available grid based infrastructures to solve scientific (or other) type of applications which need harmony among distributed hardware, software resources and human users. To achieve better usage of resources in a secure and intelligent way, scalable, reliable, and dynamic infrastructures are current requirements. Based on the current trends, we propose a secure, dynamic, and intelligent resource management and access method employing mobile agents along with the grid infrastructure.

1 Introduction

Nowadays, computer users are managing gigabytes or terabytes of data with their computation equipments. This usage contains file exchange through a network, storing them in a local storage, or processing the data based on some criteria [2]. Even though rapid enhancements in computers and related technologies are satisfy the most of the processing and storage need, some applications still require massive amounts of CPU power and/or storage areas (GNOME, satellite imaging etc.). The common concern (or limitation) is the environment in which the applications are required to run. It is open, heterogonous, and dynamic.

Distributed computing, which is a proposed solution to the aforementioned requirements, which has been used for quite some time with technology specific security implementation catering to the requirements of secure data and services, as can be seen in RMI, CORBA, DCOM technologies. Those technologies have been adopted into enterprise environments with composition of these environments in terms of users, resources, and static services.

Any distributed computing platform (including grid) needs to satisfy scalability and performance demands. What clearly distinguishes grid from these platforms is its features resulting in security requirements which cannot be addressed by existing

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 312 – 320, 2006.

security technologies for distributed platforms. Elements of grid are decided in a dynamic manner, such that the trust relationship among these elements needs to be established during application execution time.

A grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities by means of establishing, managing, and evolving multi-organizational federations. Dynamic, autonomous, and the domain independent nature of the grids allows ubiquitous access to computing, data, and services [3].

There are complexity and security challenges for grid as well as other distributed computing approaches that prevent them to be used conveniently. The autonomy and user involvement, resource access, and billing can be listed as complexity challenges whereas authentication and privacy are the main concerns for security of the system.

Although the challenges exist, the community is trying to grant a service-oriented architecture which is available anytime and anywhere in a secure way (i.e. use of intelligent agents). Distributed heterogeneous grids are considered an ideal place for establishing an agent based network for dynamic real time decision making. Agents, as autonomous entities, are capable of adapting to the environment to complete the life cycle by means of learning, interacting, communicating, and decision making. Therefore, agents are subject to metamorphism in the sense that the self learning mechanism helps the adaptation process. Furthermore, predictability of an agent serves as foresight which is a much needed tool for a dynamic distributed environment [4].

Based on expectations and concerns, our objective is to provide an environment where interactions, communications, and collaborations are conducted in a reliable, secure, and intelligent manner. In the proposed framework, mobile agent's code and state will be protected from malicious attacks by employing a simple and secure DED[1] ciphering technique. Use of such decomposition based ciphering architecture with an intelligent reasoning will result in low resource usage and sufficiently secure execution. Moreover, as fault tolerance is one of the inevitable requirements for the current distributed systems, the system allows the code or state recovery without using backups or large log files.

2 Background

Grids propose to complement and extend distributed resources by forming a dynamic collection of resources – users, machines, and/or applications – spread across multiple enterprises, continents supported by a software backbone called middleware [5]. Some of the very common attributes of grid environment are:

1. Coordinated resource sharing in dynamic, multi institutional virtual organizations.
2. Users and resources may be large, unpredictable and dynamic at any point
3. Each resource and user will have local policies and technologies that cannot be replaced by VO (Virtual Organization)

[1] DED (Decomposed Encoding and Decoding) is developed by the authors and the patent is pending.

4. Users and resources located in different organizations which do not have trust agreements
5. These organizations have different security protocols and implementations
6. Grid interactions are basically interactions between services and services acting on behalf of users

More precisely, these are the security challenges posed by the dynamic and disparate composition of grids. How remote users can access a resource in a Virtual Organization is still an unsolved security problem as VO resources and users can be part of different VOs simultaneously [6, 7].

There are several proposed approaches to handle the grid related security and resource access problems. Some of them are Cluster-on-Demand (COD) [8], Workspace Management Service [9], Virtual appliances [10], and semantic grid [11]. These approaches are mainly to create a big virtual cluster by including the remote resources and its management. The main problem with these approaches is that they are either not considering the security or rely on GSI (Grid Security Infrastructure). Another issue is the job management; some of the approaches are not using an intelligent approach to execute a job while others are only focusing how to execute it. Since the main goal is to enable the user to identify and use the fastest and cheapest available resource regardless of location, there should be an entity that can do both (resource access and job management) in cost-effective, secure, and fault tolerant way. In addition, this entity should satisfy the reservation, provisioning, scheduling requirements of the grid user.

Mobile Agents (MA), a subset of the agent theory, equipped with a profile of user wishes can achieve their objectives in a deliberate manner. Fortunately, they can also communicate with other agents without guidance of the creator. Agents have the following popular features: dynamism, intelligence, decentralization, and flexibility [12]. Combining the MA advantages with an efficient cryptosystem will result in a powerful and reliable framework that can be used by any information system which has privacy and speed as prime requirements. It can be made suitable for use in dynamic environments by employing MA. In turn, MA on grid technology achieves scalable intelligent system.

3 Design

Currently, grid and agent communities are focusing on open distributed service systems from different perspective. Grid is focusing on creating a reliable and secure resource sharing mechanism, while agents are focusing on intelligent architecture and problem solving. There is a convergence between approaches. Grid needs autonomous behavior, which can be attained by means of agents, agents requires a secure platform to run, which grid provides.

The proposed approach is a mobile agent based intelligent interface that works with the grid tools like GRAM (Grid Resource Access and Management), GridFTP, and MDS (Monitoring and Discovery Service), as shown Figure 1. Current Delegation and single sign-on features are done by using proxy certificates which have two problems. The first one is the duration of certificate which is usally effective

for 12 hours, and may not be long enough to finish the job or may be too long for a specific job which may be compromised. The second problem with the certificate is its protection. It is stored in */tmp* folder in unix machines and protected only by file protection. Since mobile agents have the predictability feature based on current knowledge, a certificate can be issued based on estimated job execution time.

Moreover mobile agent can either use contrat-net type of coordination method or take the job upon itself to the resource to execute it.

The architecture, as a resource manager, utilizes resource by means of agent server on each node considering the fact that nowadays over 70% of the CPU power is wasted or not utilized. Condor [13] like approach has an advantage which is being able to work on heterogeneous network. The heterogeneous nature of agents can coordinate the nodes to work together.

3.1 Agent Communication

A predictable coordination approach must be employed to make agents successful in a dynamic environment [14] so that they can solve problems that are not easy to handle by a single agent. In an agent environment, communication with the other agents is the way of coordination to get the job done and/or to achieve better and efficient results. Another advantage of coordination is the capability of finding a resolution, in case of conflict. Agents have to learn from the interactions and search so that they can perform more effectively.

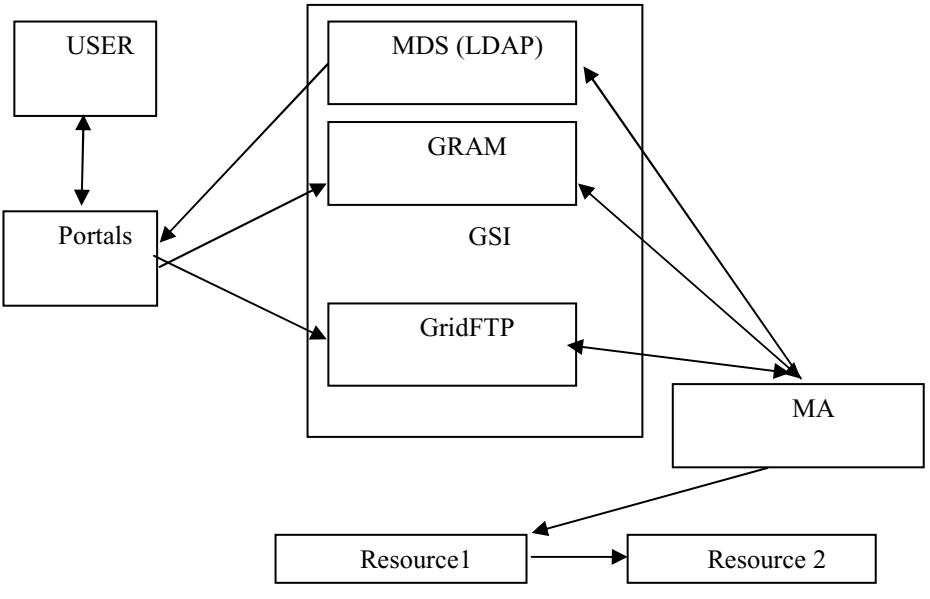


Fig. 1. The system overview

Integration of agents with grid and web services is one of the biggest challenges. The use of XML technology as a MA communication tool serves as a competitive option to overcome this challenge. There are a numerous number of communication theories for the area and, although some organizations are trying to set some standards (i.e. FIPA, OMG) over varying theories, there is still no widely accepted approach. XML is used to mark up the data to form a specification such as the Simple Object Access Protocol (SOAP) for data exchange between systems. Since XML describes the data that contains elements and attributes, it is useful for data exchange between applications and domains [15]. Due to the aforementioned advantages over the other techniques, XML is preferred as a communication tool.

A sample of an XML based message content is shown Figure 2. In this message, *Ai* is used to show the destination Agent ID (line 2), *Am* is used to show the source Agent ID (line 3), *t* is used to show the time in which the message is created (line 4), and finally the message itself (line 5).

<message ID>
<destination> *Ai*</destination>
<source> *Am*</source>
<time> *t* </time>
<message> *message body* </message>
</message ID>

Fig. 2. A sample message

Agents can interact with the other agents in the local or remote hosts for collaboration purposes. If the messaging occurs in a same host, the message is written into a shared memory. It is similar to a blackboard type of coordination model where the interaction and coordination can be achieved through an asynchronous structure.

3.2 Agent Transfer

The agent transfer is one of the fundamental issues of a Mobile Agent framework in which agent interacts with its environment for the perception and learning purposes. The way that those interactions are handled determines the robustness and the reliability [16]. A fault tolerant agent movement is a requirement for dependable distributed environment addressed in several recent researches [17].

Considering current MA threats of eavesdropping and masquerading, communication and transfer becomes an increasingly important matter. In order to overcome such threats, MA designers use several encryption/decryption techniques between sender and the receiver [16].

In our study, use of comparatively new encoding technique, DED, will yield to fail the aforementioned malicious attempts due to its unique design and lack of the right decoding information. Moreover, adapting the Agent Transfer Protocol (ATP) [18] will handle agent transform from one execution medium to another (Figure 3). When an agent request a dispatch from its current host, (GO remote(x)), the host will contact and agree upon the departure details (i.e. agent name, user) with the intended remote

server. Due to the security constraints, DED provides a mechanism to maintain the secrecy of agent code and its state.

During first departure, the agent code is encrypted using DED $e_{\pi_1}^{\ell_i}(A_i)$ with a pseudo- random function with key k_1. When the agent finishes its execution in Host A, it requests transfer to the second stop in the itenary (Host B). The code and state are encrypted using k_2 and k_1 respectively. The Host B can not access the state gained in Host A (S_A), because it does not know the k_1.

After ciphering, the agent is transferred to Host B in $e_{\pi_2}^{\ell_i}(A_i) + e_{\pi_1}^{\ell_i}(S_a)$ form. In order to employ the fault tolerance after deciphering the agent's code, mediator B deploys the checksum ($CS(A_i)$). The checksum result and k_2 are sent to the agent's Owner as message (M_1) in the ciphered form to prevent attacks on the message.

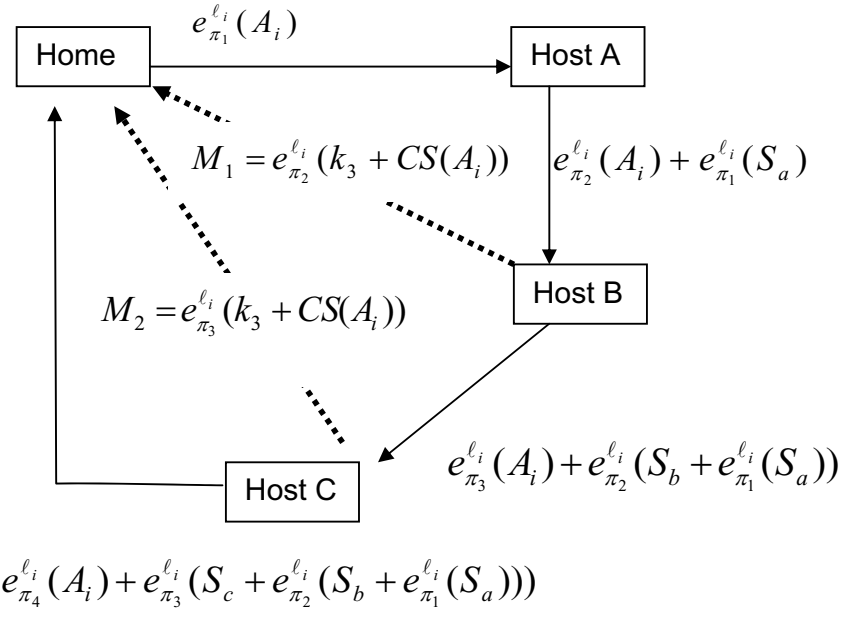


Fig. 3. ATP protocol

4 Decomposed Encoding and Decoding (DED)

Current grid technology is being used in static networks. Existing grid environments uses public key architecture (X509) for security implementations. As known, for efficiency, public key along with a symmetric algorithm is advised.

In the proposed framework DED is used, as a symmetric algorithm. DED, which prevents unauthorized receivers from accessing or understanding transmittals, is a substitution cipher with layering ability. With the 52 random letters (26 upper case and 26 lower case) assignment, provides the possibility of 8.07 x 10E67 different representation.

The formal definition of the cipher system is $\mathrm{M} = \{T, \mathrm{C}, \mathrm{K}, \ell\}$, where T is plaintext, C is ciphered text; K is key space and ℓ is layer number. In addition, it has at least two nodes: encryption and decryption.

4.1 Encryption

Definition 1: The representation of source document is defined as $T = \mathrm{Z}_{90}$, $t_i \in T$ where Z_{90} denotes a domain with 90 elements, t_i is possible plaintext which is composed of 26 upper/lower case, 10 numerals, and 28 special characters (including space), and i is communication instance.

Definition 2: The possible key space is represented as $K = {}_{52}P_{52}$, where $\pi_i \in K$ reflects the key driven from one of the Pseudo Random Number function ($\pi_i = Rnd(s_i)$) in which s_i shows seed.

Definition 3: The ciphered text symbolized as $C = \mathrm{Z}_{52}$ has a domain of 52 elements and $c_i \in C$ is member of C domain.

Based on these definitions, an initial message can be denoted as;

$$C_i = \begin{cases} \left[e_{\pi_i}^{l_i}(t_i)\right] \vee \left[e_{\pi_{i-1}}^{l_{i-1}}(s_i)\right] \vee \left[e_{\pi_{i-1}}^{l_{i-1}}(l_i)\right] & \text{for } i>0 \\ PK(s_i) & \text{for } i=0 \end{cases} \tag{1}$$

where ℓ_i is layer number, PK is Public Key, and $e_{\pi_i}^{\ell_i}(x)$ is the encoding function, which can be represented as;

$$e_{\pi_i}^{\ell_i}(t_i) = \left\{a^{\ell}(\pi_i(a^{\ell-1}(\pi_i(\ldots a^1(\pi_i(t_i)))))\right\} \tag{2}$$

The encoding function uses a previously created assignment table, which is;

$$a(\pi_i(t_i)) = (w_m) \tag{3}$$

where $w \in \mathrm{Z}_{52}$ representing the domain of 52 elements, and m represent the number of Ws between 0<m<10.

4.2 Decryption

At the receiving end, the definitions 1, 2, and 3 are valid and applicable. When the ciphered message received, the deciphering process occurs and it can be represented as;

$$t_i = \begin{cases} d_{\pi_i}^{l_i}(c_i) & \text{for } i>0 \\ PK^{-1}(s_i) & \text{for } i=0 \end{cases} \tag{4}$$

where $d_{\pi_i}^{\ell_i}(x)$ is decoding function and represented as;

$$d_{\pi_i}^{\ell_i}(c_i) = \left\{\pi_i^{-1}(a^1(\pi_i^{-1}(a^2(\ldots \pi_i^{-1}(a^{\ell}(c_i)))))\right\}$$

As in encoding function, decoding function also uses assignment table (equation (3)). Furthermore, the system has ability to break the plaintext down into several plaintexts and process different layering for each of them against Frequency Analysis attack.

5 Conclusion

In this paper, an alternative to the existing grid resource access and management methods is proposed. With the proposed approach, a confidential, autonomous, and dynamic infrastructure with efficient and cost-effective resource usage can be achieved.

The employment of such decomposition based ciphering architecture with an intelligent reasoning will result in low resource usage and secure enough execution. Moreover, as fault tolerance is one of the inevitable requirements for the current distributed systems, the system allows the code or state recovery without using backups or large log files.

References

1. G. von Laszewski, J. Gawor, P. Lane, N. Rehn, M.l Russell: Features of the Java Commodity Grid Kit. Concurrency and Computation: Practice and Experience 14(13-15): 1045-1055 (2002).
2. U. Topaloglu and C. Bayrak, “Polymorphic Compression”, The 20th International Symposium on Computer and Information Sciences, October 26-28,2005, Istanbul, Turkey.
3. I. Foster, C. Kesselman, S. Tuecke, “The Anatomy of Grid Enabling Scalable Virtual Organizations”, Intl. J. Supercomputer Applications, 2001.
4. M. Baker, R. Buyya, D. Laforenza: Grids and Grid technologies for wide-area distributed computing. Softw., Pract. Exper. 32(15): 1437-1466 (2002).

5. Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, “The Physiology of the Grid, An Open Grid Services Architecture for Distributed Systems Integration”, tech. report, Glous Project; http://www.globus.org/research/papers/ogsa.pdf
6. N. Nagaratnam, et al., “The Security Architecture for Open Grid Services”, http://www.globus.org/toolkit/security/ogsa/ggf5-presentations/OGSASec-ggf5.pdf
7. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc., ‘A Security Architecture for Computational Grids’, 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.
8. Chase, J., L. Grit, D. Irwin, J. Moore, and S. Sprenkle, Dynamic Virtual Clusters in a Grid Site Manager. accepted to the 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.
9. Keahey, K., I. Foster, T. Freeman, X. Zhang, D. Galron. Virtual Workspaces in the Grid. Europar 2005, Pisa, Italy, August, 2005.
10. Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M.S. Lam, and M. Rosenblum. Virtual Appliances for Deploying and Maintaining Software. in Proceedings of the 17th Large Installation Systems Administration Conference (LISA '03). 2003.
11. C. Goble, D. De Roure, "The Semantic Grid: Myth Busting and Bridge Building," in Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-2004), Valencia, Spain, 2004.
12. H. Endo, M. Noto, H. Toyoshima, “Quantitative Evaluation of Communication Traffic of Mobile Agents in Distributed Constraint Satisfaction Model”, International Conference on Systems, Man and Cybernetics October 10-13 2004 The Hague, The Netherlands
13. Condor project http://www.cs.wisc.edu/condor/
14. H. Nwana, L. Lee, N. Jennings, “Co-ordination in multi-agent systems”. In H. Nwana and N. Azarmi, editors, Software Agents and Soft Computing, volume 1198 of LNAI. Springer-Verlag, 1997.
15. C. Bayrak and U. Topaloglu, "Description of Free Structures with Linear Syntax," 7th World Conference in Integrated Design and Process technology, Austin, TX. December 3-6, 2003.
16. J. Ferber, “Multi-Agent systems”, Addison-Wesley, 1999.
17. J. Hartline, “Mobile Agents: A Survey of Fault Tolerance and Security”, Unpublished Manuscript.
18. F. Hohl, “An Approach to Solve the Problem of Malicious Hosts”. Institute of Parallel and Distributed High-Performance Systems (IPVR), Fakultat Informatik Universitat Stuttgart Breitwiesenstr, March 1997. Netherlands

A System Theory Approach to the Representation of Mobile Digital Controllers Agents

Fernando J. Barros

Universidade de Coimbra
Departamento de Engenharia Informática
3030 Coimbra, Portugal
barros@dei.uc.pt

Abstract. In this paper we provide a formal definition of the component concept using the Heterogeneous Flow Systems Specification formalism (HFFS) a general system theory approach to the representation of timed systems. The HFFS formalism permits the development of hierarchical and modular components whose structure evolves over time. HFSS mobile components, a particular form of structural evolution, can represent mobile entities that exist in many systems like, for example, mobile communication devices and mobile software agents. The formal description is used to model a simple control application, based on mobile software agents, where a mobile hybrid component acts as a digital controller, allowing the runtime update of the control system of a vehicle.

1 Introduction

Dynamic stricture models offer a framework for accessing the performance of systems exhibiting an evolving structure. The Heterogeneous Flow System Specification (HFSS) [1] is a set theoretic representation of dynamic structure models that supports the construction of simulation components in a hierarchical and modular form. A particular form of structural adaptation requires the displacement of a component from one network component to another one. The HFSS formalism provides the semantics for fully supporting component migration.

We present a formal description of components in the HFSS formalism. HFSS provides a unified representation of hybrid components with no distinction between mobile and non-mobile entities.

Mobile Agents (MAs) are a software technology that involves the migration of software units, called agents, among computers through a network [4], [5], [6], [7], [9]. The network is usually the Internet or a mobile phone network. Mobile Components, due to structural similarity, offer a natural way to represent Mobile Agents, and provide a framework for modeling and testing agent applications [2].

To exemplify the concepts introduced in the paper we present a simplified version of a cruise control system where vehicle velocity is set by a PI digital controller.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 321 – 333, 2006.

Considering that the vehicle is of difficult access and it is operated autonomously, the upgrade of the controller needs to be made remotely. In this case mobile agents may become a good solution.

2 Heterogeneous Flow Models

The Heterogeneous Flow System Specification (HFSS) is a formalism able to represent piecewise constant partial state components. It is based on the CFSS [6] and DEVS [10] formalisms, what makes it unified framework to represent hybrid systems.

2.1 Basic Model

A basic model in the *Heterogeneous Flow System Specification* (HFSS) is formally defined by

$$HFSS = (X, Y, P, \rho, \tau, q_0, \delta, \Lambda_c, \lambda)$$

where

$X = X_c \times X_d$ is the set of input values with
  X_c, the set of continuous input values
  X_d, the set of discrete input values
$Y = Y_c \times Y_d$ is the set of output values with
  Y_c, the set of continuous output values
  Y_d, the set of discrete output values
P is the set of partial states (p-states)
ρ: $P \rightarrow \mathbf{R}_0^+$ is the time-to-input function
τ: $P \rightarrow \mathbf{R}_0^+$ is the time-to-output function
$S = \{(p, e) | p \in P, 0 \leq e \leq \nu(p)\}$ is the state set
ν: $P \rightarrow \mathbf{R}_0^+$ is the time-to-transition function defined by
  $\nu(p) = \min\{\rho(p), \tau(p)\}$
$s_0 = (p_0, e_0) \in S$, is the initial state
δ: $S \times (X_c \times X_d^{\phi}) \rightarrow S$ is the transition function, with
  ϕ, the absence of value
  $X_d^{\phi} = X_d \cup \{\phi\}$
Λ_c: $S \rightarrow Y_c$ is the continuous output function
λ: $P \rightarrow Y_d^{\phi}$ is the partial discrete output function

The discrete output function, Λ_d: $S \rightarrow Y_d^{\phi}$, is defined by

$$\Lambda_d(p,e) = \begin{cases} \lambda(p) & \text{if } e = \tau(p) \\ \phi & \text{if } e < \tau(p) \end{cases}$$

The output function, Λ: $S \rightarrow Y_c \times Y_d^{\phi}$ is defined by

$$\Lambda(s) = (\Lambda_c(s), \Lambda_d(s))$$

In Fig. 1 are represented typical trajectories of a HFSS component. At time t_0 the component is in state (p_0, e_0) and receives a discrete input x_{d_0}.

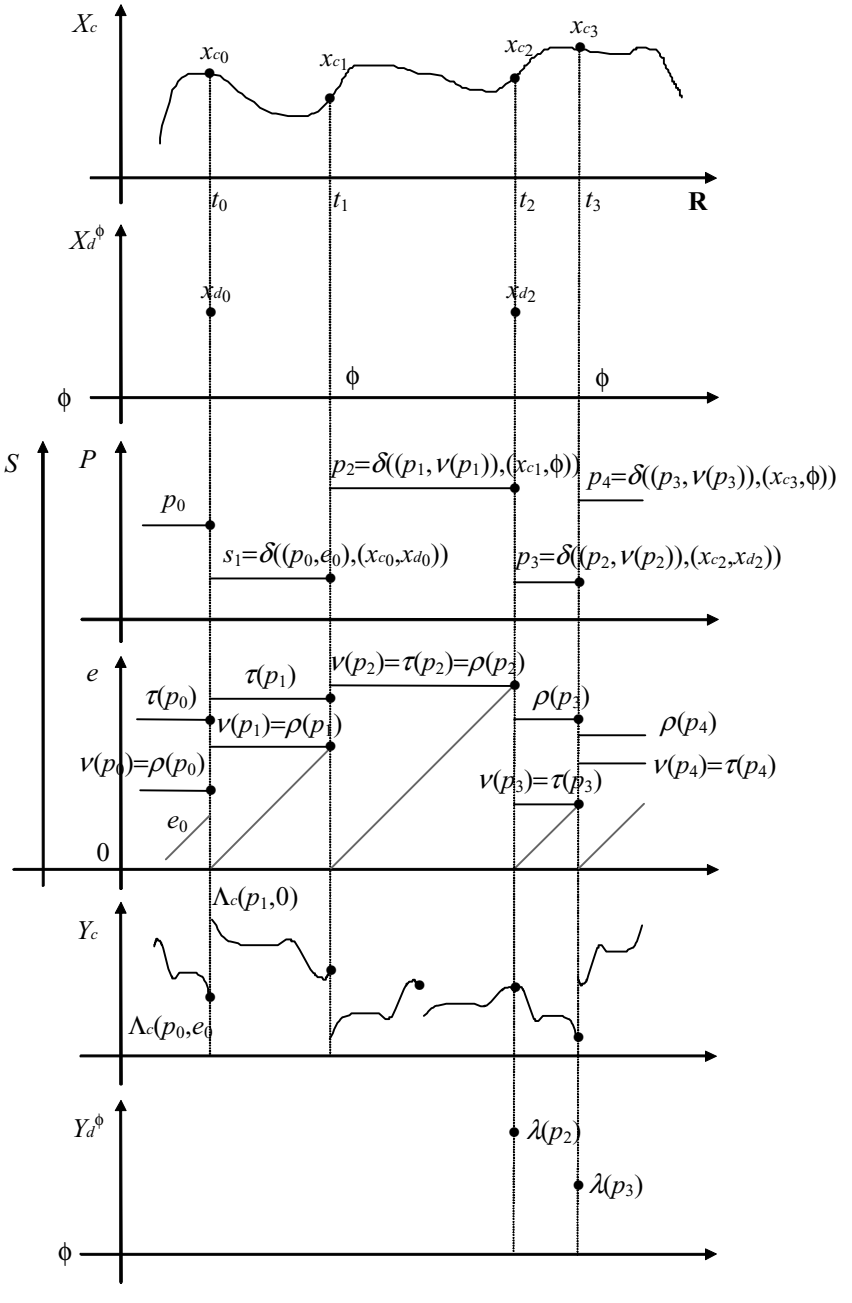


Fig. 1. HFSS trajectories

It changes then to the p-state $p_1 = \delta((p_0, e_0), (x_{c0}, x_{d0}))$. During the interval $\rho(p_1)$ no discrete input arrives, consequently, at time $t_1 = t_0 + \rho(p_1)$ the system changes to p-state $p_2 = \delta((p_1, \rho(p_1)), (x_{c1},\phi))$, where x_{c1} is the value of the continuous flow at time t_1. At p-state p_2 the time-to-input function is equal to the time-to-output function and the time to next transition is due at time $t_2 = t_1 + \nu(p_2)$. During this interval there is no discrete flow but at the end of the interval the discrete value x_{d2} arrives. The component changes at time t_2 to p-state $p_3 = \delta((p_2, \nu(p_2)), (x_{c2}, x_{d2}))$. The time-to-transition function is now equal to the time-to-input function. The component is schedule to change at time $t_3 = t_2 + \tau(p_3)$. The component changes then to p-state $s_4 = \delta((p_3, \tau(p_3)), (x_{c3}, \phi))$, for there is no discrete flow during this time interval. The continuous flow output is always defined. On the contrary the discrete flow is only non-null when the elapsed time equals the time-to-output function. In our example, this happens at times t_2 and t_3 where the discrete value is given by $\lambda(p_2)$ and $\lambda(p_3)$, respectively.

2.2 HFSS Component

HFSS components have two parts: the component model and the component model current state. The component model gives a description on how component state changes dynamically over time. This model also describes how the component state is mapped into output values. Formally, a HFSS component is defined by

$$HFSS\text{-}COMP_C = (M_C, s_C)$$

where

C is the *component name*
$M_C = (X, Y, P, \rho, \tau, s_0, \delta, \Lambda_C, \lambda)$ is the *component model*
$s_C = (p, e)$ is the *component current state*, with
$p \in P, 0 \leq e \leq \nu(p)$

Given this definition, each component has its own model, making it closer to instance based programming languages like SELF [8] rather to more classical class based languages like Java or C#.

2.3 Network Model

The HFSS formalism is based on the concept of dynamic Structure Network System [1] and it can represent models with an adaptable structure. One type of structural change supported by the HFSS formalism is the migration of a component between two models. This type of component is named here by *Mobile Component* (MC), and allows a faithful structural representation of many systems with mobile entities. The HFSS formalism can represent the key operations for enabling component mobility, namely: component removal, component transport and component restore. The formalism provides the necessary constructs to ensure that the current state (including the time spent in the current operation) of a removed component is stored so it can be latter correctly resumed.

Formally, a Heterogeneous Flow System Specification Network is a 4-tuple

$$HFN_N = (X_N, Y_N, \eta, M_\eta)$$

where

N is the network name

$X_N = X_{c_N} \times X_{d_N}$ is the set of input values, with

X_{c_N} the set of continuous input flow values

X_{d_N} the set of discrete input flow values

$Y_N = Y_{c_N} \times Y_{d_N}$ is the set of output values, with

Y_{c_N} the set of continuous output flow values

Y_{d_N} the set of discrete output flow values

η is the name of the dynamic structure network executive

M_η is the model of the executive η

The model of the executive is a modified HFSS basic model, defined by

$$M_\eta = (X_\eta, Y_\eta, P_\eta, \rho_\eta, \tau_\eta, s_{0,\eta}, \gamma, \Sigma^*, \delta_\eta, \Lambda_{c\eta}, \lambda_\eta)$$

where

Σ^* is the set of network structures

γ: $P_\eta \to \Sigma^*$ is the structure function

The network structure $\Sigma_j \in \Sigma^*$, corresponding to the p-state $p_{j,\eta} \in P_\eta$, is given by the 4-tuple

$$\Sigma_j = \gamma(p_{j,\eta}) = (D_j, \{M_{i,j}\}, \{I_{i,j}\}, \{Z_{i,j}\})$$

where

D_j is the set of component names associated with the executive p-state $p_{j,\eta}$

for all $i \in D_j$

$M_{i,j}$ is the model of component i

for all $i \in D_j \cup \{\eta, N\}$

$I_{i,j}$ is the set of components influencers of i

for all $i \in D_j \cup \{\eta\}$

$Z_{i,j}$ is the input function of component i

$Z_{N,j}$ is the network output function

These variables are subject to the following constraints for every $p_{j,\eta} \in P_\eta$:

$\eta \notin D_j, N \notin D_j, N \notin I_{N,j}$

$M_{i,j} = (X_{i,j}, Y_{i,j}, P_i, \rho_i, \tau_i, s_{0,i}, \delta_{i,j}, \Lambda_{c_{i,j}}, \lambda_{i,j})$ is a basic HFSS model, for all $i \in D_j$, with

$\delta_{i,j}$: $S_i \times X_{i,j} \to P_i$

$Z_{i,j}$: $\bigtimes_{k \in I_{ij}} V_{k,j} \to X_{i,j}$, for all $i \in D_j \cup \{\eta\}$

where

$$V_{k,j} = \begin{cases} Y_{k,j} & \text{if } k \neq N \\ X_N & \text{if } k = N \end{cases}$$

The network output function is given by

$Z_{N,j}$: $\bigtimes_{k \in I_{N,j}} Y_{k,j} \to Y_N$

Changes in network structure include the ability to change network composition and coupling. Structural changes are triggered by the transition function of the executive that can made arbitrary modifications in the network structure.

3 Cruise Control

The ability to update software systems while running is crucial in many applications. Some systems, like nuclear plants work in non-stop operation mode and software updates needs to be done while the system is running. The ability to make those updates remotely is mandatory in some applications, specially in software systems installed on devices operating in space, like satellites and probes, where upgrades needs to be done by radio communication.

We describe the skeleton of the model of a control application that uses a mobile agent to upgrade the software control part of a remotely operating vehicle.

3.1 Mobile Component

The network S, depicted in Fig. 2 represents the whole system that includes the vehicle V and a digital controller π. When the network executive receives a mobile controller it replaces the current controller by the new one. The vehicle sends information about current velocity v and mass m. The controller receives the error $v - v_R$, vehicle mass and a discrete flow stop signal , and sets vehicle thrust th.

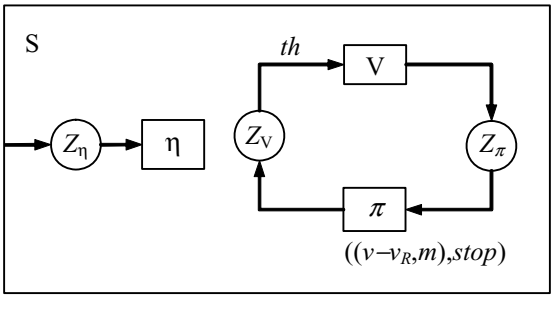


Fig. 2. Digital controller and vehicle

The network S represented by

$$HFN_s = (X_s, Y_s, \eta, M_\eta)$$

where

$X_s = \{\} \times \mathbf{\Pi}$, where $\mathbf{\Pi}$ represents the set of models of PI digital controllers. Models of the set $\mathbf{\Pi}$ share the same partial states, making them compatible.

$Y_s = \{\} \times \{\}$

The model of the executive is given by

$$M_\eta = (X_\eta, Y_\eta, P_\eta, \rho_\eta, \tau_\eta, s_{0,\eta}, \gamma, \Sigma^*, \delta_\eta, \Lambda_{c\eta}, \lambda_\eta)$$

where

$X_\eta = X_{c\eta} \times X_{d\eta} = \{\} \times \mathbf{\Pi}$
$Y_\eta = Y_{c\eta} \times Y_{d\eta} = \{\} \times \{\}$
$P_\eta = \{\pi_0, \pi_1, \pi_2, \ldots\}$
$\rho_\eta(p) = \tau_\eta(p) = \infty$
$s_{0,\eta} = (\pi_0, 0)$
$\Sigma^* = \{\Sigma_0, \Sigma_1, \Sigma_2, \ldots\}$

To simplify the notation we use M_i as the model of controller π_i instead of M_{π_i}.

$\gamma(\pi_i) = \Sigma_i = ($
$\{\pi_i, \mathrm{V}\},$
$\{M_i, \mathrm{M_V}\},$
$\{I_\eta = \{S\}, I_\mathrm{V} = \{\pi_i\}, I_{\pi_i} = \{\mathrm{V}\}, I_\mathrm{S} = \{\ \}\}$
$\{Z_\mathrm{V}((v, m), x_d) = ((v - v_R, m), x_d),$
$Z_\mathrm{S}() = (\phi, \phi),$
$Z_\eta(x) = (x),$
$Z_{\pi_i}(x) = (x)$
$\}$
$)$

Let us consider that the component to be update is named π_i and is defined by $C_i = (M_i, s_i)$. When the executive receives a new model $M_j = (X_j, Y_j, P_j, \rho_j, \tau_j, s_{0,j}, \delta_j, \Lambda_{c_j}, \lambda_j)$ the transition function is defined by

$$\delta_\eta(\pi_i, e, (\phi, M_j)) = \pi_j$$

where the model of π_j is given by

$$M_j^\partial = (X_j, Y_j, P_j, \rho_j, \tau_j, s_i, \delta_j, \Lambda_{c_j}, \lambda_j)$$

subject to the constraint $s_i \in S_j$

The difference between the received model M_j and the update model M_j^∂ is on the initial state. The ability to use the state of the current model as the initial state of the new controller permits to avoid glitches in the control signal and to receive the input sample at the same instant, like if the controller was not updated. The state trajectories of the two components at the update time are depicted in Fig. 3. At time t a new controller model M_j arrives at the vehicle system. Vehicle executive replaces the current controller by the new one. The initial state of the new controller is the same as the current state of the removed one. The next transition of the new controller occurs at time $t + \rho_i(p_i)$ as it would be if the controller was not updated. This behavior is imposed by the fixed sampling rate assumption.

The network S has an unbounded number of structures since its can accommodate an infinite number of different PI controllers. These controllers are not generally known in advance given that can be developed only after the system is operating. Given that the whole set of potential controllers are unknown in advance, traditional representations based on switching systems are not adequate.

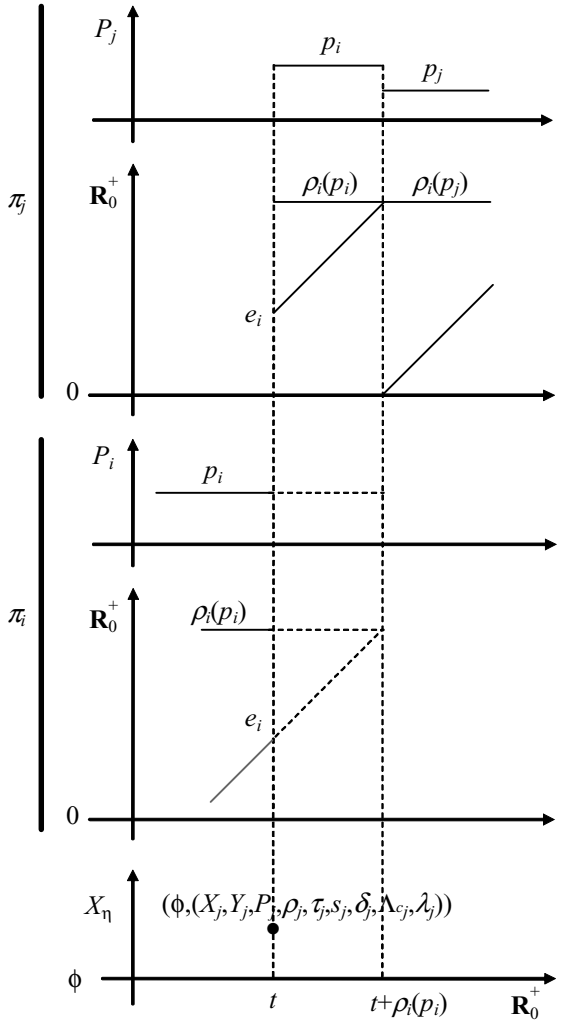


Fig. 3. State trajectories of the controllers

3.2 Vehicle Model

The vehicle is represented by component V that exposes vehicle velocity and mass. A discrete event signal is also produced by detector Δ when fuel tank is empty. This information is used to stop the controller. The vehicle structure is depicted in Fig 4. Vehicle acceleration a depends on the thrust *th* set by the controller, drag proportional to vehicle velocity *v*, and vehicle mass *m*. Fuel consummation is proportional to thrust. The differential equations that model vehicle mass, velocity and position are solved by HFSS components specialized in numerical integration.

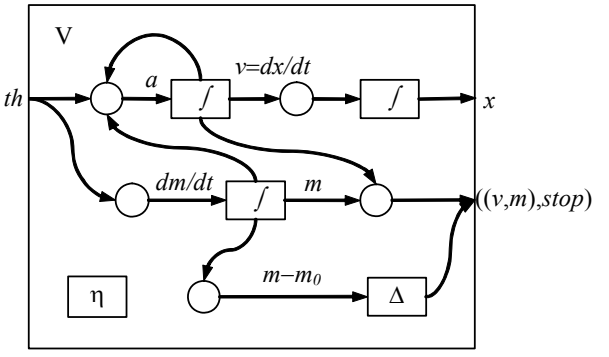


Fig. 4. Vehicle dynamics

The acceleration is given by

$$a = d^2x/dt^2 = (th - k_f\, dx/dt) / m$$

Mass consummation is given by

$$dm/dt = -k_m\, |th|$$

The event detector Δ signals when the vehicle mass is equal to m_0 corresponding to an empty fuel tank. This event will trigger a stop signal to the controller that will set the thrust to zero.

3.3 The Controller Model

The controller tries to keep vehicle velocity to a constant value. Control actions depend on both velocity error and error integral, i.e., a PI digital controller type was chosen. The current controller has fixed values for the P and I parameters. Since the vehicle mass diminishes as fuel got consumed, the possibility to change the controller enables controller upgrade by more effective algorithms that can adjust PI parameters. The HFSS model of the initial PI digital controller π_0 is given by

$$M_0 = (X, Y, P, \rho, \tau, s_0, \delta, \Lambda_c, \lambda)$$

where

$X = \mathbb{R}^2 \times \{\text{stop}\}$

$Y = \mathbb{R} \times \mathbb{R}$

$S = \{\text{run, out, stop}\} \times \mathbb{R}^5$

$\rho(phase, \alpha, \beta, int, err_{-1}, out) = \alpha$

$\tau(phase, \alpha, \beta, int, err_{-1}, out) = \beta$
$s_0 = ((\text{run}, 0, \infty, 0, 0, 0), 0)$
$\delta(((\text{out}, \alpha, \beta, int, err_{-1}, out), e), (\_, \text{stop})) = (\text{stop}, \infty, \infty, 0, 0, 0)$
$\delta(((\text{run}, \alpha, \beta, int, err_{-1}, out), e), ((err, m), \phi))) = (\text{out}, \infty, 0, int_1, err, out)$
with
$int_1 = int + e \times (err + err_{-1}) / 2$
$out = \max\{0, 10\ err + 0.5\ int_1\}$
$\delta(((\text{out}, \alpha, \beta, int, err_{-1}, out), e), \_) = (\text{run}, 2, \infty, int, err_{-1}, out)$
$\Lambda_c((phase, \alpha, \beta, int, err_{-1}, out), e) = out$
$\lambda(phase, \alpha, \beta, int, err_{-1}, out) = out$

We consider that sampling is done at a fixed rate of 2 s. However, the HFSS formalism permits to represent more flexible controllers based on variable rate sampling. Proportional and integral parameters have a value of 10 and 0.5, respectively. The minimum value of the control action is zero, meaning that not break action is applied.

After studying vehicle performance it was decided to use an improved controller that adjusts proportional and integral parameters taking into consideration vehicle mass *m*. The new controller π^{∂} is described by

$$\pi^{\partial} = (X, Y, P, \rho, \tau, s_0^{\partial}, \delta^{\partial}, \Lambda_c, \lambda)$$

where
$s_0^{\partial} = s_{\pi_0}$
with s_{π_0}, the current state of controller π_0
$\delta^{\partial}(((\text{out}, \alpha, \beta, int, err_{-1}, out), e), (\_, \text{stop})) = (\text{stop}, \infty, \infty, 0, 0, 0)$
$\delta^{\partial}(((\text{run}, \alpha, \beta, int, err_{-1}, out), e), ((err, m), \phi))) = (\text{out}, \infty, 0, int_1, err, out)$
with
$int_1 = int + e \times (err + err_{-1}) / 2$
$out = \max\{0, m\ (10\ err + 0.5\ int_1)/300\}$
$\delta^{\partial}(((\text{out}, \alpha, \beta, int, err_{-1}, out), e), \_) = (\text{run}, 2, \infty, int, err_{-1}, out)$

In order to replace the controller by the new one without glitches we use the current state of controller π_0 as the initial state of the new controller π^{∂}.

3.4 Simulation Results

Simulation results for vehicle thrust, velocity and mass are presented in Fig. 5, Fig. 6 and Fig. 7, respectively. Two curves represent vehicle thrust, velocity and mass under controllers π_0 and π^{∂}. The third curve represents vehicle behaviour when controller π^{∂} replaces controller π_0 after 20 s of system operation. Numerical results were produced in the CAOSTALK modeling and simulation environment [1]. System parameters are set to: v_R = 30 m/s, k_f = 0.75 Kg/s, k_m = 0.02 Kg/N and m_0 = 100 Kg. Values are illustrative and do not intend to represent an existing system.

As show in this example, HFSS components can model complex systems requiring a heterogeneous representation requiring differential equations, digital controllers and hybrid mobile agents. Agents can be used to provide component update as required in systems that cannot be easily accessed.

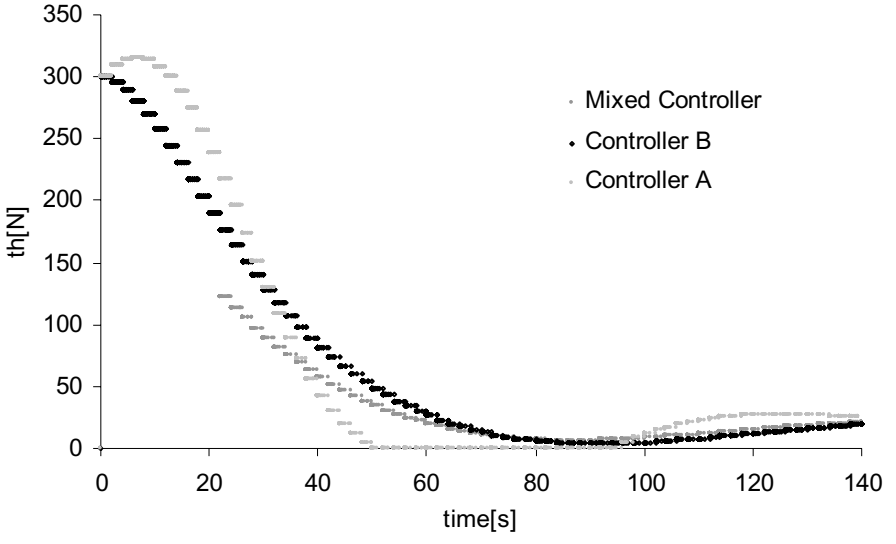


Fig. 5. Vehicle thrust

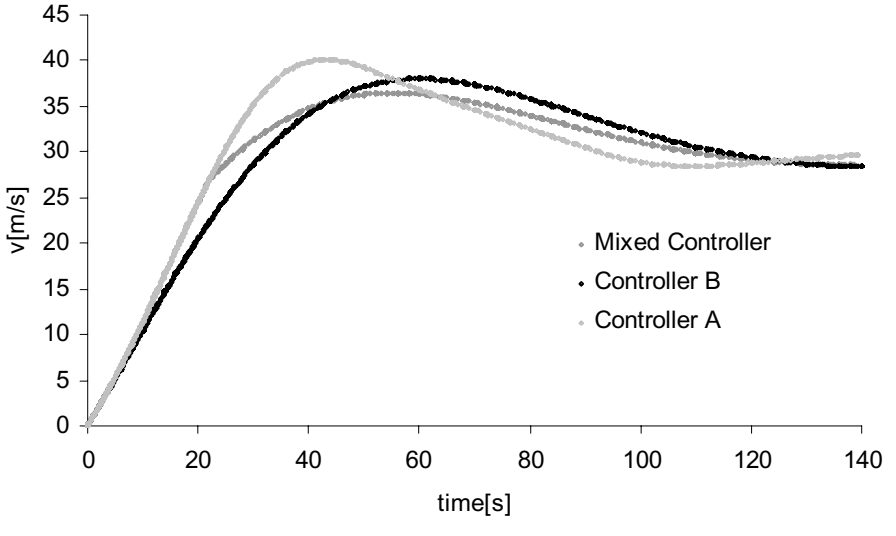


Fig. 6. Vehicle velocity

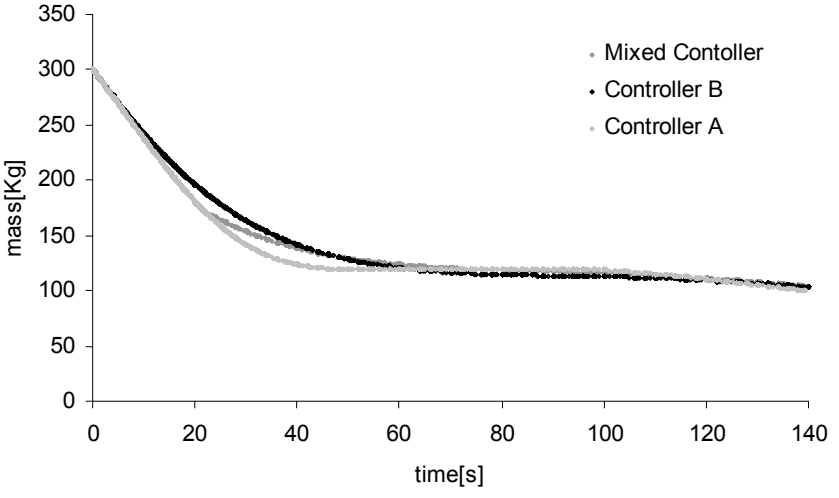


Fig. 7. Vehicle mass

4 Conclusions

The HFSS formalism can describe simulation components as independent units that can be built by hierarchical composition. This formalism can represent heterogeneous systems at both basic and network level. HFSS networks have a dynamic structure nature that can be modified by adding and removing components and interconnections. Structure adaptation includes the ability to represent mobile components, components that can migrate from one network to another while maintaining their state. Due to structural similarities, mobile components offer a framework for modeling and simulation of mobile agent systems. The simulation of a system that uses mobile agents to represent digital controllers was presented showing the HFSS formalism ability to model hybrid mobile systems.

Acknowledgements

This work was partially funded by the Portuguese Foundation for Science and Technology (FCT), under project POSI/SRI/ 41601/2001.

References

1. Barros, F.J. Modeling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 4, pp. 501-515, 1997.
2. Barros, F.J. DSDE: A Formalism for Representing Mobile Components. In *Agent Oriented Approaches in Distributed Modeling and Simulation: Challenges and Methodologies*, Dagsthul Seminar Report 245, pp. 9, 1999.

3. Barros, F.J. Towards a Theory of Continuous Flow Models. *International Journal of General Systems*, pp. 29-39, 2002.ï
4. Cardelli, L. and A.D. Gordon. Mobile Ambients. In *Foundations of Software Science and Computation Structures*, LNCS, Vol. 1378, pp. 140-155, 1998.
5. Fuggetta, A., Picco, G. P., and Vigna, G. Understanding Mobile Code. *IEEE Transactions on Software Engineering*, Vol. 24, No. 5, pp. 342-361, 1998.
6. Harrison, C.G., D.M. Chess and A. Kershenbaum. Mobile Agents: Are they a Good Idea? *IBM Research Report*, 1994.
7. Lange, D.B. and Oshima, M. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesly, 1998.
8. Unger, B. and Smith, R. SELF: The Power of Simplicity. Technical Report TR-94-30, Sun Microsystems Lab, 1994.
9. White, J. Mobile Agents White Paper. *http://www.klynch.com/documents/ agents*, 1996.
10. Zeigler, B.P. *Theory of Modeling and Simulation*. Academic Press, 1976.

Towards Adaptive Migration Strategies for Mobile Agents

Steffen Kern[1] and Peter Braun[2]

[1] Friedrich Schiller University Jena, Computer Science Department
Ernst-Abbe-Platz 2, 07743 Jena, Germany
steffen.kern@informatik.uni-jena.de
[2] Swinburne University of Technology, Faculty of Information Technologies
John St, Hawthorn Victoria 3122, Australia
pbraun@ict.swin.edu.au

Abstract. Mobile agents were introduced as a new design paradigm for distributed systems. One advantage of mobile agents is to reduce network traffic as compared to the client-server paradigm, simply by moving code close to the data instead of moving large amount of data to the client. Unfortunately, many mobile agent toolkit suffer from to simple migration techniques. Therefore, we argue in this paper for an new migration technique that supports an adaptive decision on the code and data relocation technique. We propose several techniques for code analysis and alteration and present an algorithm to determine the optimal migration strategy within this model under the assumption of full knowledge about the application and network environment.

1 Introduction

Mobile agents have been introduced as a design paradigm for distributed applications. A mobile agent is a program that can migrate from host to host in a network of heterogeneous computer systems and fulfill a task specified by its owner. It works autonomously and communicates with other agents and host systems. During the self-initiated migration, the agent carries its code and some kind of execution state with it.

In current mobile agent toolkits (we restrict ourselves to Java toolkits in the following), such as Jade [3] or Aglets [12], code and data relocation is implemented in a fixed way, which might lead to poor migration performance in specific application scenarios compared with client-server based approaches.

We have developed a new mobility model, named Kalong [6], which provides fine-grained migration strategies. Whereas in other mobile agent toolkits, code can only be transmitted either by pushing all classes to the destination agency or loading all classes from a code server on demand, an agent in Kalong can decide for each class how it should be transmitted. If the agent already knows that a specific class will not be used at the next destination agency, the agent opts against class transmission. Otherwise, if the class is very likely to be used at the next agency, the agent can enforce to transmit it as part of the migration process.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 334–345, 2006.

In [5] we have presented an approach based on static program analysis. In this paper we present our approach to increase migration performance by enabling agents to adapt their code and data relocation strategy during run-time based on past experiences.

The remainder of this paper is organized as follows. Section 2 gives a short introduction into current migration strategies found in actual mobile agent toolkits and *Kalong*, our own mobility model, is described in section 3. We present some techniques for migration optimization in section 4 and our migration planing algorithm in section 5. An evaluation of that algorithm is shown in section 6. Section 7 discusses related work and Section 8 concludes this paper.

2 Current Migration Strategies

In the following, we use the term *migration strategy* to describe the code and data relocation strategy. Regarding data relocation, almost all current toolkits implement a common strategy based on the following framework. When a mobile agent decides to migrate to another host, the underlying agency (agent server) is responsible to stop agent execution, serialize the agent (and all referenced objects, i.e. the object closure), and to transfer the serialized agent (i.e. the execution state) in one shot to the destination agency. The destination agency receives and deserializes the agent and finally re-starts it by creating a new thread and invoking a specific method. None of the available agent toolkits allows to send only parts of the state or to download other parts, for example rarely used data items, later.

Regarding code relocation, current toolkits implement different techniques, which can be roughly classified into *pull* and *push* strategies. The first strategy does not transmit any code along state transmission and uses Java's built-in technique to download classes on demand from the agent's home agency. Pull strategies are used for example in Grasshopper [2], and Jade [3].

The second class of code relocation strategies is named *push* strategies. The code of an agent (together with the code of all referenced objects, i.e. the class closure) is transmitted at once to the next destination. This strategy corresponds to one important characteristics of mobile agents, that is autonomy: The agent does not need a connection to its home agency after it has been started.

It is no surprise that neither pure push nor pure pull strategies lead to an optimal network load and application performance in all cases. For example, pulling classes during run-time from a distant home agency might be very expensive (and unreliable) and pushing classes might cause superfluous class transmission, since some classes (or parts of them) are never used.

3 The Kalong Mobility Model

The thesis of our work is that mobile agents suffer from fixed migration strategies as they are provided in today's agent toolkits. In our opinion, mobile agents should not only be able to select a specific migration strategy from a catalog

of available strategies before run-time but also able to continuously adapt this strategy during run-time.

Our approach is based on a new mobility model, named Kalong, which is the result of an in-depth analyzes of mobile agent's drawbacks compared with client-server approaches [6]. With Kalong, the migration strategy is not fixed but can be *programmed* by the agent during run-time for each migration. Kalong differs from current mobility models in three main aspects:

1. Kalong defines a new agent representation and new transmission units. In our model, mobile agents not only consist of an object state and their code, but have also an *external state*, which comprises of data items that are not part of the object state. A mobile agent's code is no longer transmitted in form of classes or JAR files, but in a new transmission format that we call *code unit*. A code unit comprises of at least one class or many classes, which are supposed to migrate together.
2. Kalong defines two new agency types, namely a *code server* agency, from which an agent can download code on demand, and a *mirror* agency, which is an exact copy of an agent's home agency. A mobile agent can define an agency to be a code server or mirror and later release it again dynamically during run-time.
3. Kalong defines a new class cache mechanism. Our class cache is able to avoid transmission of identical classes used in different agents and can distinguish between different versions of the same class. The class cache does not only prevent unnecessary class download (pull) but also unnecessary class transmission when used in push strategies.

This mobility model is implemented as an independent software component, which can be considered as a virtual machine for agent migration. Basic commands of this virtual machine are, for example, sending an agent's state or sending an agent's code unit. Classes that were not sent during migration must be loaded (pulled) on demand later. This may result in any kind of mixture of push and pull strategies. Additionally, the migration strategy also defines, which data items of the external state should be migrated to the next destination or, for example, sent back to the agent's home agency. In none of the available mobile agent toolkits, an agent can influence the migration strategy during run-time. It should be obvious that by using these primitives for state and code transmission, it is possible to describe all migration strategies that we have introduced in the previous section. A detailed description of Kalong can be found in [6].

4 Techniques for Migration Optimization

Current migration techniques transfer Java code either on the level of classes or class packages (JAR files). We propose to transmit code on the level of methods, because we learned from our experience of implementing small to medium mobile agent-based applications that methods of one class may not have equal or

similar execution probabilities. This section presents some techniques to identify methods with similar execution probabilities and afterwards alter the agents code according to the preceding results.

4.1 Code Analysis

Static Analysis. All static analysis steps and the class split process described in the next chapter use *ByCAl* (Byte Code Analyzer), a Java API developed at Friedrich Schiller University to read, change, and write Java byte code.

The analysis process starts with with a control flow analysis followed by the construction of an intra- and interprocedural *control flow graph* [15] which will be the starting point for all further analysis steps. Afterwards a *call graph* is built whose vertices represent methods with at least one entry point and whose edges designate a possible method call. During our upcoming analysis steps each vertex will be labeled with the execution probability of the method it represents and each edge `(a,b)` with the probability that `b` is called by `a`. To construct this *call graph* we use a *class hierarchy analysis* followed by a *rapid type analysis* to reduce the number of edges in the graph. See [9, 7] for details. Later, based on this *call graph*, we will determine the decomposition of the processed agent. We now perform a *data flow analysis* [16] and *value range propagation* (VRP) [13] to determine the execution probabilities of each branch following a conditional jump. To execute a fast *value range propagation* the control flow graph is transformed into *static single assignment form* [8]. For branches where VRP is not able to determine any probability information, several heuristics are applied [1].

A static analysis has several drawbacks. The slow execution is a minor problem as the analysis must only done once. The bigger problem is the reliability of the results. The analysis only considers the agents classes but we think that good execution probabilities can only be derive by analysing the agent and the underlying agency. But this would significantly increase running time or would not be possible at all due to hardware limitations (memory, etc.).

Dynamic Profiling. Because of the drawbacks of static analysis we implemented another technique, namely *dynamic profiling*. We assume an agent (or, in general, different agent instances of the same agent type) to execute the same task frequently (not necessarily the same itinerary), so that we can infer from class usage information gathered during one (sample) execution to succeeding executions. Dynamic profiling is implemented by annotating Java byte code with a counter for each class (a counter is a class variable), which is increased for every method invocation. Because class variables are not part of the serialized object closure, the agent must save these values before a migration to the object state. After one agent execution, the agent knows how often which of its classes has been used and it can decide on classes to push and those classes to be pulled during a subsequent execution.

As a small example for dynamic profiling, consider the following setup. We have an agent containing four different methods which are used during its tour. The agent will visit three remote agencies calling method 1 on agency 1, method

2 on agency 3 and method 3 on agency 3. Method 4 is called on every agency. Additionally, depending on a random value, each method may be called once more on each agency. Thus leads to at most two executions of each of those four methods. In our test, the agent repeated this tour 500 times.

The results are shown below. Table 1 displays the absolute execution counter values and Table 2 show the derived execution probabilities for each pair of method and agency. As can be seen, execution probabilities of 1.0 match exactly to the test parameter and all other values lie inside the range we defined for the random value.

Table 1. Counter Values from Profiling

	Agency 1	Agency 2	Agency 3	$\sum$
Method 1	666	162	128	956
Method 2	226	674	198	1098
Method 3	270	247	734	1251
Method 4	841	770	781	2392

Table 2. Derived Execution Probabilities

	Agency 1	Agency 2	Agency 3
Method 1	1.0	0.324	0.256
Method 2	0.452	1.0	0.396
Method 3	0.54	0.494	1.0
Method 4	1.0	1.0	1.0

4.2 Class Splitting

Based on the execution probabilities derived from code analysis we split Java classes on the level of byte code into smaller transmission units by a technique named *class splitting* [10], which is done transparently to the agent and agent programmer before an agent is started on its home agency. Class splitting is based on standard techniques known from compiler construction and include sophisticated program analyzes and static profiling techniques.

The general process of splitting a class file is rather simple, the problems are hidden in the details. The process could be divided into two steps:

1. Moving original methods into a new class file and ensure that they are still operational in this new environment. The new classes will be in the same package as the original class.
2. Create a stub method replacing the original method. This stub has to behave in exactly the same way as the original method i.e. it must have the same number of arguments, same return type and same flags.

Original Agent

```
class Agent implements Runnable, Serializable {
  //used on every agency
  public void run() { ... }
  //used at home agency
  private void startAgent() { ... }

}
```

Splitted Agent

```
class Agent implements Runnable, Serializable {
  private Agent$s01 s01;
  public void run() { ... }
  //stub method to forward request
  private void startAgent() {
    if(s01 == null) s01 = new Agent$s01();
    s01.startAgent();
  }
}

//new class with "home" methods
class Agent$s01 implements Serializable {
  private Agent this$0;
  private Agent$s01() { ... }
  private void startAgent() { ... }
}
```

Fig. 1. Class Splitting Example

As can be seen methods like startAgent or back which are only executed at an agents home agency are moved into a new class thus reducing the size of the original class. During the split process a stub method for each moved method was added to the original class which will forward a request to the method to the new class. Although this example shows Java source code class splitting will be performed at byte code level.

As useful as class splitting seems to be in cases of monolithic large agents – especially on networks with low bandwidth – there are several situations where splitting should be omitted. As it is easy to see moving a method into a new class is useless if the size of the stub method is large as the size of the original method. Another situation occurs when a relatively small agent moves in a network with high bandwidth. The speedup which can be achieved by making this agent even smaller should be nearly not measurable. A third is the worst case when the complete agent code is needed on each remote agency which means that after splitting all classes are send to each remote agency. This would producing a higher network load than the single original class because the sum of the class sizes of the splitted agent is bigger than the size of the original class.

5 Migration Planing

After an agent's code has been split and annotated, the agent is started and migrates along a predefined itinerary. Actually, it is not compulsory for our approach to have the complete itinerary predefined. However, the more destinations are known in advance, the more accurate are the results of the migration planner.

Our network performance model for planning the migration strategy consists of an agent itinerary of m agencies $\mathcal{L} = \{L_1, \ldots, L_\mathrm{m}\}$, which have to be visited in a fixed order and to execute some tasks on each server. We consider L_1 to be the home agency. We assume an agent to consist of u units of code $\{U_k\}$, each of length $B_\mathrm{c}^k, k = 1, \ldots, u$. The agent carries a state of length B_{state} and d external

data items $\{D_k\}$, each of length $B_{\mathrm{d}}^k, k = 1 \ldots, d$. The usage probability of code unit k on server L_{i} is $PU_{L_{\mathrm{i}}}^k$ and the usage probability of data item k on server L_{i} is $PD_{L_{\mathrm{i}}}^k$. Before an agent has been executed, all probabilities are assumed to equal 0.5 and after each round, probabilities are set according to the results of the dynamic profiling.

In case that a code unit must be loaded on demand, a request of length B_{cr} (the same for all units B_{c}^k) must be sent; in case of a missing data item, a request of length B_{dr} must be sent. For each pair of servers we assume to know network link throughput $\tau : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$ and link delay $\delta : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$ in advance. We assume neither τ nor δ to be symmetric.

We define A as the set of all possible actions defined by Kalong. An action is any basic command of Kalong, for example to send a code unit, send a data item, send the agent state, send a request to load a data item or code unit, etc. We define costs in terms of network load and processing time for each Kalong action. For example, to send code unit U_k from server L_{i} to server L_{j} produces a network load of B_{c}^k bytes and takes $\tau(L_{\mathrm{i}}, L_{\mathrm{j}})^{-1} B_{\mathrm{c}}^k + \delta(L_{\mathrm{i}}, L_{\mathrm{j}})$ sec.

We can also determine the costs for necessary code and data item downloading operation. For example, if the agent is located at server L_{i} at the moment and data item D_k has to be loaded from the agent's home agency with a specific probability, this produces a network load of $PD_{L_{\mathrm{i}}}^k (B_{\mathrm{dr}} + B_{\mathrm{d}}^k)$ bytes and takes $PD_{L_{\mathrm{i}}}^k ((\tau(L_{\mathrm{i}}, L_1)^{-1} B_{\mathrm{dr}} + \delta(L_{\mathrm{i}}, L_1)) + (\tau(L_1, L_{\mathrm{i}})^{-1} B_{\mathrm{d}}^k + \delta(L_1, L_{\mathrm{i}})))$ sec.

A migration strategy $MS : \mathcal{L} \to \mathcal{P}(A)$ defines for each location L_{i} the set of actions $\{A_k\}$ to be executed at this location in order to let the agent migrate to $L_{\mathrm{i}+1}$.

To determine the costs (execution time) for a specific migration strategy MS, we have to consider for all locations the time for migration T_M (i.e. executing all Kalong actions) and the time for executing the agent T_E (which might include necessary code and data downloading):

$$T(MS) = \sum_{L_{\mathrm{i}} \in \mathcal{L}} \left(\sum_{A_j \in MS(L_{\mathrm{i}})} T_M(L_{\mathrm{i}}, A_j) + T_E(L_{\mathrm{i}}) \right)$$

In order to determine the optimal migration strategy, i.e. the migration strategy with minimal total cost, we map it to the problem of finding the shortest path in a directed graph. Each node represents a state S, which is defined as a mapping of code units and data items to servers, from which they can be downloaded:

$$S = \{\{U_k\} \to \mathcal{P}(\{L_{\mathrm{i}}\}), \{D_k\} \to L_{\mathrm{i}}\}$$

Edges represent agent migrations (Kalong actions) or agent executions and each edge can be weighted with the associated cost of either migration or execution. The process of constructing this graph is outlined in Fig. 2.

This algorithm is configured with information about the agent (itinerary, class size, data item size, usage probability of each code unit and data item, etc.) and information about the network environment (link bandwidth, and latency). The

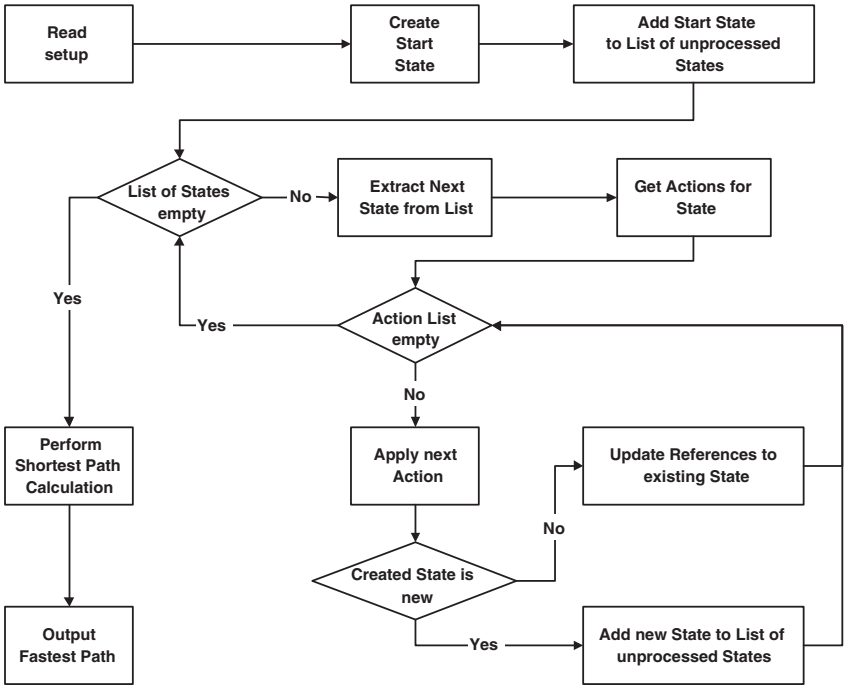


Fig. 2. Outline of the migration planning algorithm

output of the planning algorithm is a sequence of commands for the Kalong virtual machine that leads to optimal migration behavior in terms of network load and processing time.

6 Evaluation

We conducted several experiments with the migration planer and we will present one of the most interesting ones. We call this scenario *Nomad User* as the agent is started from a mobile device with a slow and unstable connection to the network. After leaving its home agency the agent will visit four remote agencies which reside in standard LAN. Bandwidth is 100.0 MBit/sec, latency 0.5 sec and availability 85.0 % for all but the connections to the home agency. Those connections have a latency of 5.5 sec and availability is 5.0 %. For the usage probabilities of the code units take a look at table 3. The agent's size and that of each code unit is 10.0 KB.

In this scenario the installation of a code server or mirror should increase round trip time because loading classes from any server but the home agency will be much faster than pulling them from home. And we wanted to guide the

Table 3. Usage Probabilities of Code Units

	Home	Agency 2	Agency 3	Agency 4	Agency 5	Agency 6
CodeUnit 1	100.0	0.0	0.0	0.0	0.0	100.0
CodeUnit 2	0.0	100.0	0.0	0.0	0.0	100.0
CodeUnit 3	0.0	0.0	100.0	0.0	0.0	100.0
CodeUnit 4	0.0	0.0	0.0	100.0	0.0	100.0
CodeUnit 5	0.0	0.0	0.0	0.0	100.0	100.0

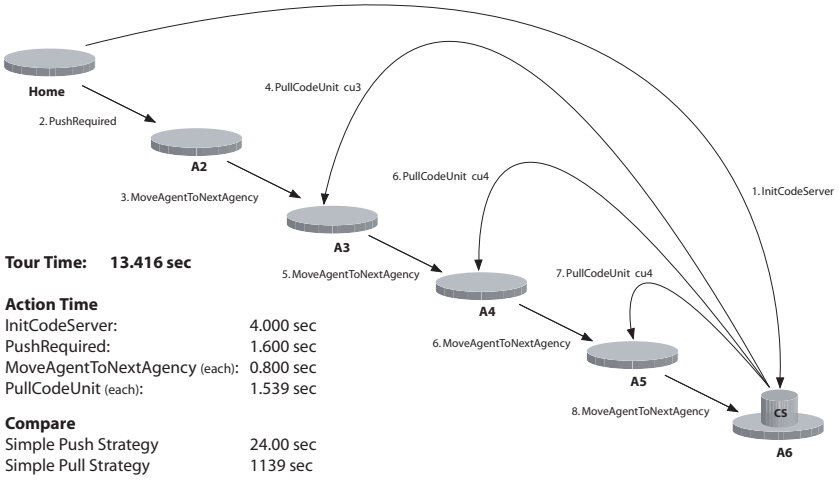


Fig. 3. Nomad User Example

planer in the decision which agency should become code server/mirror with the fact that all code units are needed at the last agency.

Figure 3 shows the fastest sequence of migration actions for this scenario.

The two most important points to notice are the installation of a code server and the location of that code server. The experiment led to exactly that kind of migration strategy we had in mind in advance. A utilization of a code server yielded to a faster round trip time and the location was chosen to be the agency needing the most code units.

A simple *Push* strategy took 24.0 sec. Compared to that result we are 44% faster. The *Pull* strategy led to a roundtrip time of 1139.44 sec, what is an utterly devastating result. This is caused by the fact that each code unit must be loaded from home agency via that very unstable and slow connection. The fastest strategy using a code server on agency 6 is thus 98% faster.

Other migration plans considered the installation of more than one code server and/or mirror, but the overall running time does not benefit from such a step. The tour time for such plans is bigger than 14 sec.

7 Related Work

To our knowledge the only paper dealing with class file splitting as a way to reduce network load of mobile code is by Krintz et al. [11]. However there are some differences between their and our approach. First of all they are dealing with Java Applets which are a special kind of mobile code. An applet is send to a remote server, executed there and dies afterwards. According to the list of method types we gave in section 3 an applet only contains methods of type 2 and 4 which greatly limits the number of optimization strategies compared to mobile agents.

The main goal of the authors is to reduce the startup time of an application especially if the application is an applet which is started after all classes have been transfered to the remote server. To achieve this goal the authors propose a technique which moves methods which are not used during the startup phase or which are rarely used to a new class. This new class is then loaded by another thread while the main class is already in use. Compared to our approach they only split classes into two parts - a hot one for the first used methods and a second for the later on used methods, whereas we create several cold parts (or split classes as we call these parts). The disadvantage of the Krintz approach is that if only on rarely used method is needed the complete cold part must be transfered. On the other hand, in our approach each new class file creates an overhead to the total size of all new class files. Thus, moving each method into a single class would also be ineffective.

An interesting point in their paper is class prefetching which is used to reduce delays if a cold class has to be loaded. They examine the control-flow of a class and determine the point at which they know for certain that class x is needed sooner or later. They start a new thread at this point to load class x in parallel. The Kalong mobility model also allows class prefetching but unfortunately, prefetching statements increase code size so additional research is necessary if this technique can be used to optimize our approach.

Although Krintz' paper describes class splitting only on source code level, we have to assume that they also work with byte code. Otherwise they could not determine when it is beneficial to split a class file since the relevant file size is the size of the class file which contains Java byte code and not the size of the source code file. However, their paper mainly focuses on the prefetching process and describes the splitting process only concise. Thus, we are unable to compare their splitting process with ours.

To summarize, the Krintz approach allows a faster starting and response time, but they do not optimize for network bandwidth. At last, even more bytes must be transfered, because the sum of the size of all new class files is higher than the original size and the added statements for prefetching also increase the size.

8 Conclusion and Outlook

In this paper we have presented our approach for adaptive mobile agents in order to increase migration performance. We have presented a new mobility model,

Kalong, which enables mobile agents to adapt their migration strategy during run-time. We have presented an algorithm to determine a migration strategy under the assumption of full knowledge of the itinerary, network parameters, and execution probabilities of the agents. These execution probabilities are determined from dynamic profiling, which is implemented by annotating Java byte code before agents' execution.

At the moment, we are conducting experiments in wide-area network to evaluate the quality of our approach. To forecast network performance we are working on techniques comparable to [14] in combination with our domain manager concept [4] implemented in Tracy.

In the longer run, we work on the integration of these concepts into a framework for adaptive migration of mobile agents, including an adaptive decision between remote communication and migration.

References

1. Thomas Ball and James R. Larus. Branch prediction for free. In *Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation*, pages 300–313. ACM Press, 1993.
2. Christoph Bäumer, Markus Breugst, Sang Choy, and Thomas Magedanz. Grasshopper — A universal agent platform based on OMG MASIF and FIPA standards. In Ahmed Karmouch and Roger Impey, editors, *Mobile Agents for Telecommunication Applications, Proceedings of the First International Workshop (MATA 1999), Ottawa (Canada), October 1999*, pages 1–18. World Scientific Pub., 1999.
3. Fabio Bellifimine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade – A White Paper. *EXP in search of innovation*, 3(3):6–19, 2003.
4. Peter Braun, Jan Eismann, and Wilhelm R. Rossak. Managing Tracy Agent Server Networks. Technical Report 12/01, Friedrich-Schiller-Universität Jena, Institut für Informatik, June 2001.
5. Peter Braun and Steffen Kern. Towards adaptive migration techniques for mobile agents. In Zahia Guessoum, editor, *Fifth Workshop on Adaptive Agents and Multi-Agent Systems (AAMAS 2005), Paris (France), March 2005*, 2005.
6. Peter Braun and Wilhelm R. Rossak. *Mobile Agents–Basic Concept, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers, 2005.
7. Craig Chambers, David Grove, Greg DeFouw, and Jeffrey Dean. Call graph construction in object-oriented languages. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA-97),*, volume 32, 10 of *ACM SIGPLAN Notices*, pages 108–124. ACM Press, 1997.
8. Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Trans. Program. Lang. Syst.*, 13(4):451–490, 1991.
9. David Grove and Craig Chambers. A framework for call graph construction algorithms. *ACM Transactions on Programming Languages and Systems*, 23(6):685–746, November 2001.

10. Steffen Kern, Peter Braun, Christian Fensch, and Wilhelm R. Rossak. Class splitting as a method to reduce the migration overhead of mobile agents. In Robert Meersman, Zahir Tari, and Angelo Corsaro, editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004, Agia Napa (Cyprus), October 2004, Proceedings, Part II*, volume 3291 of *Lecture Notes in Computer Science*, pages 1358–1374. Springer Verlag, 2004.
11. Chandra Krintz, Brad Calder, and Urs Hölzle. Reducing transfer delay using java class file splitting and prefetching. *ACM Sigplan Notices*, 34(10):276–291, 1999.
12. Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.
13. Jason R. C. Patterson. Accurate static branch prediction by value range propagation. In *Proceedings of the ACM SIGPLAN 1995 conference on Programming language design and implementation*, pages 67–78. ACM Press, 1995.
14. Wolfgang Theilmann and Kurt Rothermel. Dynamic distance maps of the internet. In *Proceedings IEEE INFOCOM 2000, The Conference on Computer Communications, Volume 1, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Reaching the Promised Land of Communications, Tel Aviv (Israel), March 2000*, pages 275–284. IEEE Computer Society Press, 2000.
15. Jiangjun Zhao. Analyzing Control Flow in Java Bytecode. In *In Proceedings of the 16th Conference of Japan Society for Software Science and Technology*, pages 313–316, 1999.
16. Hans Zima and Barbara Chapman. *Supercompilers for parallel and vector computers*. Addison-Wesley, 1991.

Agent Modeling of Tetrahedron-Based Structures

Charles Sebens[1] and Walt Truszkowski[2]

[1] MIT
77 Massachusetts Avenue
Cambridge, MA 02139-4307
csebens@mit.edu
[2] NASA Goddard Space Flight Center
Code 588
Greenbelt, MD 20771
USA
walt.truszkowski@nasa.gov

Abstract. Goddard Space Flight Center (GSFC) is working on developing new kinds of robotic structures capable of: goal-oriented motion, changing its form to optimize its function, adapting to new environmental demands, and/or repairing itself. A series of increasingly complex roving shapes leading up to the tetrahedron were considered. This paper addresses this innovative use of multi agent system technology that is being used to achieve the desired autonomous behaviors.

1 Introduction

ANTS (Autonomous Nano Technology Swarm) SMART (Super Miniaturized Addressable Reconfigurable Technology) architectures were initiated at the Goddard Space Flight Center (GSFC) to develop new kinds of robotic structures capable of: goal-oriented motion, changing its form to optimize its function, adapting to new environmental demands, and/or repairing itself.

To begin to explore the possibilities of these concepts and the possible application of multi-agent control, a series of increasingly complex roving shapes leading up to the tetrahedron were considered. The goal is to have the structure move from an initial location to a specified goal location. The tetrahedron has six struts that can be reversibly deployed or stowed. More complex structures will be formed by interconnecting these reconfigurable tetrahedra, making structures that are scalable and massively parallel systems. The full functionality of such a complex system requires fully autonomous operations [1].

As illustrated in Figure 1, the tetrahedron (tet) has four nodes and six expandable struts. It "walks" by extending certain struts, changing its center of gravity and "falling" in the desired direction. Currently, the basic structure, the tetrahedron, is being modeled as a cooperating/collaborating 4-agent system with an agent located on each node of the tet. (An agent, in this context, is an intelligent autonomous process capable of deliberative and reactive behaviors as well as social and introspective behaviors.) The desired result is the realization of a truly autonomous tetrahedron structure.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 346 – 353, 2006.

This paper will address this innovative use of multi-agent system technology that is being used to achieve the desired autonomous behaviors as well as the increasingly complex staged approach in the development of the multi-agent tet.

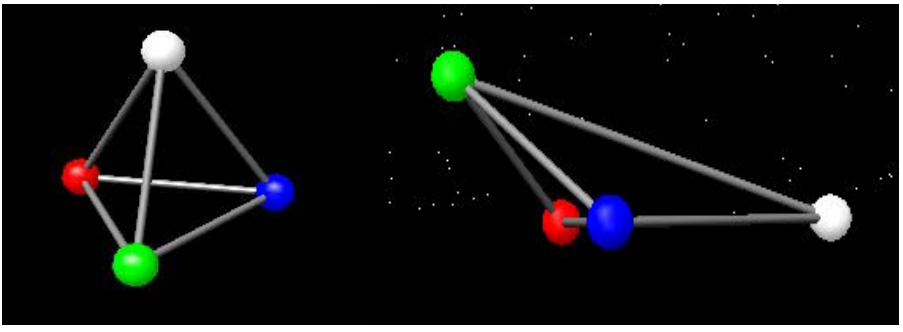


Fig. 1. Structure of a tet, and possible configuration of a fallen tet

2 Problem

This on-going research project centers on designing a constrained multi-agent system that efficiently and effectively allows a simple tetrahedral rover (a rover consisting of a single tetrahedron) to move autonomously on a surface and maneuver past obstacles it may encounter (such as a rock, wall, incline, cliff, or ceiling) in order to achieve a specified goal. Each simple tetrahedral rover will be autonomously controlled by a group of four agents acting in a coordinating and collaborative fashion. The agents, located at the four corners, or "nodes," of the tetrahedron, will control motion by specifying the extension of the struts that are connected to the nodes of the tetrahedron. The structs, connecting the agents, gives rise to the "constrained" multi-agent system. A design objective is that the struts must be extended in a way that maximizes distance covered while minimizing power usage. The agents also must be able to evaluate and maneuver past obstacles they may encounter as they travel to the goal.

3 Approach

In order to achieve the overall objective of realizing a fully autonomous simple tetrahedral walker, a phased approach is being followed. For this phasing, the simple tetrahedral walker was considered through four increasingly complex systems, as illustrated in Figure 2: the node (a), the inchworm (b), the triangle (c) and the tetrahedron (d).

Underlying Hypothesis: We have identified two types of emergence: emergence of autonomous behaviors in a simple tetrahedral structure arising from the progressive composition of autonomous behaviors realized by going systematically from a dot to an inchworm to a triangle to the simple tetrahedron. This is the major type of

emergence hoped for in our current research. The emergent autonomous behaviors of more complex tetrahedral structures, realized from the composition of the individual autonomous behaviors of the basic building blocks (the tetrahedral), is the second type of emergence (not addressed in this paper). Please note that the "emergence" of more complex autonomous behaviors from simpler autonomous behaviors (both at the simple and complex structures levels) is a major hypothesis of the current research.

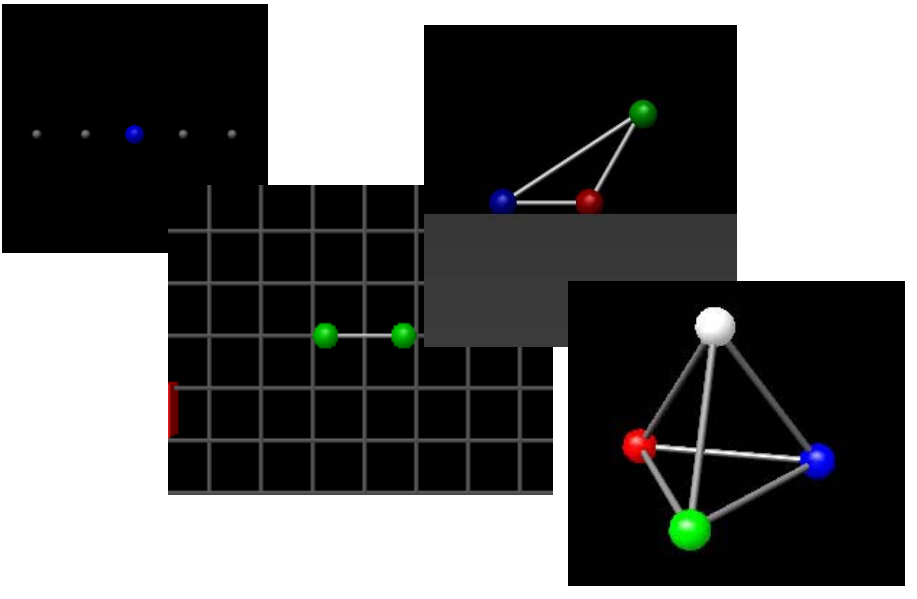


Fig. 2. (a) node, (b) inchworm, (c) triangle, (d) tetrahedron

4 Components of the Tet-Walker

The basic idea that underlies our current work is that if the progressively more-complex behaviors of the sub-structures of the tetrahedron are simulated and then analyzed then an analyses of the simulation code will help in understanding the "reasoning " that is being followed. This reasoning will be embedded in the agents associated with the tet-walker.

Before we proceed with a discussion of the substructures associated with the simple Tet-walker we define what we mean by "agent".

> Agent - "a physical or virtual entity which is capable of acting in an environment, which can communicate directly with other agents, which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize), which posses resources of its own, which is capable of perceiving its environment (but to a limited extent), which has only a partial representation of its environment (and perhaps none at all), which possesses skills and can offer services,

> which may be able to reproduce itself, whose behavior tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and the communications it receives." [2]

4.1 The Dot

The most basic autonomous, mission-capable unit of the tetrahedron is a node (dot). To begin to develop the AI for the system, it is important to be able to design AI for the most basic part, a single node. In this scenario the capabilities of linear motion in a one-dimensional environment are assumed. Also assumed is that the agent can accept the datum of the goal (target destination, in this case specified by an x coordinate). Through top-down design the AI for the agent was then designed so that its necessary abilities could be established. Next, the program was created through bottom-up system design. The program was divided into two classes: the visualization/realization, and the AI. The visualization/realization class, hereafter referred to as the VR, simulated the physical laws of the environment and displayed an image of the scenario. The VR also acted as the messenger to the user accepting the goal location, and passed this information on to the AI as well as relayed the "thoughts" of the AI to the screen. The AI class evaluated the mission and commanded the VR class to move the dot in a certain manner. Both classes communicated, but only in methods that would approximate the assumed abilities of the rover. To evaluate the success of the program, its ability to reach a point within a specified margin of error of a target destination was evaluated.

The AI for the dot is relatively simple. Initially the dot knows the location of itself and the goal, as defined by x coordinates. The dot then identifies which side of itself the goal is on (or if it is at the goal) and takes one step in that direction. It then receives an update of its location. It uses this information to check if it has reached the goal (or is within a certain radius of it), choose a direction, and move. This loop is repeated until the node decides it has reached the goal.

This combination of traits makes the dot a simple agent with one exception, it lacks the ability to communicate with other agents (this is accepted though due to the lack of other agents in the dot's environment). It acts in the environment by moving along the line. It is driven by the goal of reaching a particular point. It possesses resources in the form of a method used to interpret goal location and output direction of motion. It is capable of perceiving its environment by accepting the location of the goal. Its behavior tends towards satisfying its objective of reaching the goal.

4.2 The Inchworm

The next logical step determined important enough to implement was the inchworm. This is the name given to two nodes connected by a strut. This problem begins to address some of the more difficult problems of the tetrahedron. These problems involve strut-based motion and communication between agents, particularly concerning agreement on strut control. The objective of the inchworm is to get either of its nodes within a specified distance; thereby making the target a circle. In this case it was assumed that:

a) motion is performed on a 2d surface,
b) circular obstacles are placed at random on that plane such that they are not on top of the inchworm or too close to the target destination at the beginning of the simulation,
c) contact with an obstacle results in mission failure,
d) either node of the inchworm can increase friction such that the strut can be extended without that node slipping,
e) both agents can receive a target destination (specified by x and y coordinates),
f) a complete view of the surroundings is available to both agents,
g) the mission is considered successful if either one of the nodes is within a specified radius of the goal,
h) both nodes have the ability to order strut extension or contraction, i) the nodes can communicate with each other, and
j) both nodes know the maximum and minimum lengths of the strut.

This program also used the VR and AI classes. The VR of this case differed from the single node in that it also communicated conversations of the agents to the user. To evaluate success of the system, goal-completion ability's dependency on the amount of obstacles was reviewed. Through multiple simulations (obstacles being placed at random), capabilities of the inchworm were evaluated.

The AI becomes far more complex in the Inchworm. Each inchworm node is "born" with knowledge of the minimum and maximum lengths of the strut connecting the two nodes, the length of one "step" (a unit used throughout loops in the program to increase speed and decrease accuracy, approaching 0, in this case, is equal to 0.0007), the three dimensional location of itself, the identity of the other agent, the identity of the virtual universe which constitutes its environment, the location of the goal, and an ID number used to identify itself in output. After all these variables have been initialized, it checks to see if it is closest to the goal (it gets the location of the other node by asking, it does not store this information.). If it is closest it sets the value of a boolean (true/false) variable "front" to true, and proceeds to control the motion. If it is not true, it sets the value to false and waits for further communication. If they are both equally close to the goal, the node with an ID number of 1 sets front to true and continues. The front node then checks to see if the goal has been reached by itself (as defined by the node being within a certain radius of the node) and requests that the other node does the same. If either node determines the goal has been reached, that node stops reasoning and tells the other to stop also. Then, a loop begins. The following sequence of steps is repeated until it is determined that the node is either at the goal, or no longer closest to the goal: First, the node checks whether it needs to align itself with the goal (if the goal is on the same line as the nodes) and proceeds to do so in the most efficient method if necessary. This step is repeated in order to allow a moving goal. The environment then updates its position. It then takes a step towards the goal (the back node remains stationary while the strut extends to the maximum (or if the node is close to the goal, some intermediate) length, after that, the strut contracts while the front node remains stationary. The positions of the nodes are then updated. Next, the node checks to see if it is still the front node. If it is not, it relinquishes control, and the other node begins this loop. If

it is, it proceeds. Finally the node checks to see if the goal has been reached, and tells the other node to do the same.

4.3 The Triangle

In order to better understand the problems of reactive motion and node adaptability in tumbling motion, a triangular configuration has been considered. Three agents located on the corners control the triangle's motion by changing the lengths of the struts connected to the corner on which they are located. The triangle moves in a linear manner, reaching a target along the line defined by an x-coordinate. The triangle moves by extending struts so that its center of mass passed the front base node (see Figure 3).

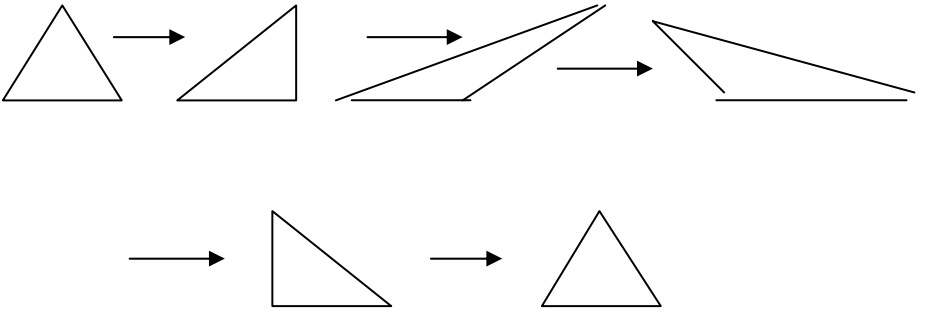


Fig. 3. Movement of the triangle

An important goal for the triangle was not only for the system to successfully move, but also for it to do so in the most efficient way possible. In this scenario, efficiency is measured by absolute value of change in length in all struts during motion. Initially, an equation was determined to find the most efficient motion. This proved too difficult to solve analytically, so instead, the simulation was run with all different possible ratios of extensions for each of the sides relative to each other. The data was then graphed in a three-dimensional graphing program developed for this data. It was extremely clear from this experiment that the most efficient motion involved maximum extension of the "far-top" strut, minimum but positive extension of the "top-close" strut, and negative extension of the base strut. There were limitations to this approach. For example, sometimes it is possible that a strut will contract and extend at different times, particularly when the first step is taken from an equilateral triangle but the following ones are taken from a fallen state. Despite these concerns, the agents were originally designed from this data.

Later, these issues were revisited and in order to combat these problems, the purely mathematical approach was reconsidered and proved to be solvable through development of a short program. In this analysis the length of one side (the base) was kept constant to eliminate inconsistencies in size. The data was then graphed in the same program and it was found that a 2.02:1.01:1 ratio of *lengths* would be most efficient for motion an infinitely long flat surface. In fact the extension should be

slightly more, because that is approximately an equilibrium position, not one that would induce falling. This data is not extremely precise (it is accurate to the 0.05), as it was just a development towards the tetrahedron.

Soon after this approach gave such helpful results, the triangle program was turned on its head, well, at least slightly tilted. It was determined that a useful ability for the triangle to have, and a good development towards the tetrahedral walker would be the ability to handle inclines. The mathematical analysis was then expanded to not just consider various length combinations, but also various angles. The most efficient motions at each of 16 different angles were then isolated. Sixth degree polynomials were fitted to the lengths of each side and given to the agents to use in controlling the triangle.

In this model, each agent is identical in processes followed, but differ in state of knowledge. The triangle nodes are instantiated with knowledge of the minimum and maximum lengths of the struts, the length of one "step", the three dimensional location of itself, the identity of the other agents, the identity of the virtual universe, the location of the goal, the angle of the incline, and an ID number used to identify itself in output. Once instantiated, the node proceeds to check if it has reached the goal. It then calculates it's relative position to the other nodes and the goal. It is either: "top" (y coordinate > 0), "far" (y = 0, distance to goal is greater than "close"), "close" (y = 0, distance to goal is less than "far"). After this, it repeats the following steps until the value of "found" is true. If it is "far," it tells the base strut to contract to the smallest length allowed. If the agent is on top, it tells the strut connecting itself to "far" to extend until it reaches a value specified by the previously determined equation (using the angle as input). If it is "far," it tells the strut going to the top node to move towards the value given by its equation. After executing its extension, the agent requests an update of its location from the environment and then uses that to determine again its relative position. Once the goal is reached, this loop ends and by communication between nodes is ended for all other agents; the mission is deemed successful.

4.4 The Tetrahedron

Initially the tetrahedron was considered without obstacles or inclines (roving on a flat surface towards an (x,z) goal). In this case each agent followed the following logic:

First, the node checks to see if it is part of the base (if the y-value of its location is equal to zero). If this condition is satisfied, the agent goes on to see which base node it is. If its location is farthest from the goal, then it tells the strut connecting itself to the top node, to extend to a set length (between two and three times the minimum strut extension). Also, it tells the two other struts it is connected to, to contract to the minimum length for a strut. If two nodes are both equidistant and farthest from the goal, one is selected at random to extend to this length, the other acts like the following nodes. If a node satisfies the first condition, but not the second, it informs the strut that connects itself to the top node to extend to a different set value (just a little more than the minimum). The extension is done in a step-based manner and is halted when the tet falls over (the top node is constantly checking to see if the tet has fallen over, or, reached the goal). Positions are reevaluated upon falling, motion is stopped when the top node has reached the goal. This motion succeeded in reaching the goal.

5 Conclusion

This paper has presented a brief overview of the current work which is going on in the attempt to realize an autonomous simple tet-walker. Much more needs to be done.

In the immediate future there are plans for tests to be run that are similar to those run on the triangle. This will output the best motion for the tet both on flat ground and inclines. Using these results the base nodes will be able to handle inclines and react appropriately while the top node manages map making and path planning.

Acknowledgements

The authors wish to acknowledge the support and encouragement supplied by Dr. Steve Curtis, Dr. Mike Rilee, Dr. Cynthia Cheung and Matt Brandt all at the Goddard Space Flight Center. Additional acknowledgements go to the directors of the internship program at the Eleanor Roosevelt High School which provided support for Chip Sebens.

References

1. The ANTS website: http://www.gsfc.nasa.gov/ants
2. Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, Addison-Wesley, 1999.

Congestion Control in Multi-Agent Systems Through Dynamic Games of Deterrence

Michel Rudnianski[1] and Hélène Bestougeff[2]

[1] University of Reims, ARESAD, 39 bis Avenue Paul Doumer, 75116 Paris –France
Michel.Rudnianski@wanadoo.fr
[2] University Paris VII – Denis Diderot
helene.bestougeff@ufr-info-p7.jussieu.fr

Abstract. Congestion control in MAS is addressed through a network of agents communicating with each other, for issuing and responding to assistance requests. The network state is analyzed as a particular qualitative game, called Game of Deterrence. A congestion avoidance algorithm is proposed, on the basis of the game solutions. The relation between the type of game reflecting the structure of the agent network, and the occurrence of congestion is then analyzed, highlighting the impact of the network size, and leading to the alternative of either implementing the algorithm, or breaking down the global network in regional networks inside which the congestion occurrence probability is low. The method is then extended to the dynamic case through an example showing existence of cycles.

1 Introduction

Multi-agent systems can be considered as a paradigm for representing workflow issues in a given organization. Workflow control, and more specifically congestion avoidance, is of utmost importance, since the latter is the basis for rationalizing the organization. Now, congestion avoidance is quite a complex and multi-facet problem. In previous papers [1,2,4,10,11] we have followed a general approach based on the generic principle of task limitation within the organizational network. More specifically, modeling the organizational network as a two-player game of deterrence, we have developed a congestion control algorithm triggered by the game solutions. The case of fuzzy games [10] has been considered, and some specific properties concerning particular graphs associated with the game of deterrence have been shown. Moreover, a categorization of the states of the game has been made on the basis of whether they would or would not lead to congestion at the next state [11].

The present paper, will extend the previous results by introducing dynamic games of deterrence, in order to look for long term solutions. By long term solutions, we here mean solutions that are satisficing in the long run, that is solutions which enable to avoid congestion over a given set of time.

More precisely, we shall introduce multi-stage games of deterrence, in which evolution of the state over time defines a trajectory, while the game target is a sub-space of the game space.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 354 – 369, 2006.

Congestion is then defined with respect to a game target and forbidden transitions from state to state.

In section 2, we shall recall the main definitions and properties of static games of deterrence. In section 3 we shall analyze as an example, how the problem of information sharing between two agents can be modeled through games of deterrence. Section 4 will first introduce the basic definitions and properties of dynamic games of deterrence, before extending the case study of section 2 to a three-agent multi-stage game of deterrence. Section 5 will consider the congestion control problem can be dealt with through games of deterrence. In particular, an analysis of the game will be made in accordance with the risk of congestion occurrence.

Last, section 6 will apply the concepts and properties introduced to long run congestion control, through developing an example.

2 Static Games of Deterrence

2.1 The Purpose of Games of Deterrence

While standard Game Theory has dramatically progressed in the study of equilibria refinements, thus trying to solve the problem of the multiplicity of equilibria, many situations occur where one is unable to precisely assess the states of the world corresponding to the selection of a particular strategy by each player. The literature offers various approaches to deal with this problem. Most of them amount to ordering the outcomes either directly, like in the usual ordinal games, or indirectly, through mapping the set of possible outcomes corresponding to a given state of the world, with a probability distribution.

Games of deterrence propose an alternative approach, based on the observation, that even if decision makers are unable to precisely assess their outcomes, more often than not, they are still able to say whether a given state of the world is acceptable (for them and / or for the other players) or not. In other words, many decision situations can be represented by analyzing states of the world with regard to a predefined threshold.

If a state of the world is considered being beyond the threshold (i.e. acceptable), the associated outcome will be 1, while on the opposite, if the state of the world is considered being beside the threshold (i.e; unacceptable), the associated outcome will be 0. This amounts to a coarse ordering of the state space.

So games of deterrence will look only for satisficing equilibria, that is, equilibria such that each player may reach an acceptable outcome. Conversely, this approach will aim as much as possible at deleting strategies that do not enable the player who selects them to get an acceptable outcome: Therefore, the term deterrence, and the fact that what is sought in these games is playable strategies.

2.2 Theoretical Model

We here recall the basic definitions and properties presented in [10]

Let R and C be two players with respective strategic sets S_R

(card S_R = n) and S_C (card S_C = p).

We shall consider finite bi-matrix games (S_R, S_C, A, B,*S*) in normal form where possible outcomes are taken from the set $\{0,1\}$. More precisely, for any strategic pair $(r,c) \in S_R x S_C$, $A(r,c) = 1$ defines an acceptable outcome for the player R,

while $A(r,c) = 0$ is not. Similar definitions apply by analogy to player C with matrix B.

A strategy r of Row is said to be *safe* iff $c \in S_C$, $A(r,c)=1$.

A non-safe strategy is said to be *dangerous*.

Let J(r) be an index called *index of positive playability*, such that:

If r is *safe* then $J(r) = 1$

If not,

$$J(r) = \prod_{c \in S_C} [1-(1-A(r,c))J(c)](1-\tilde{j}_{C_i})\hat{j}_R$$

With $\hat{j}_R = \prod_{r \in S_R} (1- J(r))$; $\hat{j}_C = \prod_{c \in S_C} (1- J(c))$

If $J(r) = 1$, strategy $r \in S_R$ is said to be *positively playable*.

If there are no positively playable strategies in S_R, that is $\hat{j}(r) = 1$, strategy $r \in S_R$ is said to be *playable by default.*

Similar definitions apply by analogy to strategies c of C .

A strategy $s \in S = S_R \in S_C$ is *playable* iff it is *either positively playable or playable by default.*

The system ***S*** of all J(r), $r \in S_R$, and J(c), $c \in S_C$, is called the *playability system of the game*.

A *solution* of ***S*** is a consistent set

$$\{ J(r_1), J(r_2),..., J(r_n), J(c_1), J(c_2)...., J(c_p)\}.$$

In the general case, there is no uniqueness of the solution.

A strategic pair $(r,c) \in S_R x S_C$ is said to be an *equilibrium* of this game if both strategies are playable for some solution of the playability system.

Given a strategic pair $(r,c) \in S_R x S_C$, r is termed *deterrent strategy vis-à-vis c* iff:

(1) r is playable

(2) $B(r,c) = 0$

(3) $c_i \in S_C$, such that $J(c_i) = 1$.

It has been shown [6] that a strategy $c \in S_C$ is playable iff there is no strategy $r \in S_R$ deterrent vis-à-vis c. Thus, the study of deterrence properties amounts to analyzing the strategies playability properties.

2.3 Graphs of Deterrence

Given a game of deterrence (S_R, S_C, A, B,*S*), we shall call *graph of deterrence*, a bi-partite graph G on $S_R x S_C$ such that, given $(r,c) \in S_R \times S_C$, there is an arc of origin r (resp. c) and extremity c (resp. r), iff $B(r,c) = 0$ (resp. $A(r,c)=0$).

Solving the playability system ***S*** amounts to determining playabilities of the graph vertices. Since a graph can be decomposed into paths and circuits, we shall call:

- *R-path* (resp. *C-path*) a path the root of which is an element of S_R (resp. of S_C);
- *Primary circuit*, a circuit such that none of its vertices has an ancestor that does not belong itself to a circuit;
- *β-graph*, a graph, that includes neither an R-path nor an C-path.

It is shown (ibid) that:

(i) if G is an R-path (resp. C-path), the only positively playable strategy for R (resp. C) is the root, while all strategies of C (resp. of R) are playable by default;
(ii) if G is a primary circuit, all strategies of both players are playable by default;

Moreover, it is shown (ibid) that through appropriate cuts, it is always possible to break down the graph of deterrence into connected parts, each one being an R-path, an C-path, or a β-graph. Hence, depending on the presence of these elementary components in the graph, one can distinguish 7 types of games : type R, type C, type β, type R-C, type R-β, type C-β, type R-C-β.

(1) if G is an R-path, the only positively playable strategy for R is the root, while all strategies of C are playable by default;
(2) if G is a primary circuit, all strategies of both players are playable by default;
(3) if G is a β-*graph*, a solution of *S* satisfies:

(i) for any strategy s_0,

$$J(s_0) = \neg[\hat{j}_R \vee \hat{j}_E] \wedge \forall\, s \in N(s_0)\colon J(s) \wedge \forall s' \in N'(s_0)\colon [\neg J(s')]$$

where $N(s_0)$ (resp. $N'(s_0)$) is the set of the first strategies met when following G backward from s_0, and belonging to the same strategic set as s_0 (resp. to the other);

(ii) on a path, the vertices positive playability is determined by the parity of their distance to the first strategy met when following G backwards ;

(iii) each player has at least one non positively playable strategy.

This typology leads in turn to the Classification Theorem (ibid):

(1) Given a game of deterrence, its playability system's solution set is not empty.
(2) The game type defines the solution set.

It follows from (1) that every game of deterrence has an equilibrium, but the above shows that this equilibrium may not be unique.

With a small number of additional concepts, all previous definitions and results are extended to the case of N-player games [8].

We shall illustrate these concepts by an elementary example, in which we shall show the relation between a standard game and games of deterrence.

3 Example 1

Rupert and Charlene who work for the same company, are involved in a supposedly collaborative process during which they may exchange information.

Providing information takes time and has therefore a cost (data mining, synthesis, shaping, transmission etc.). We shall first assume that the aggregate cost of providing a unit of information is 2, while the gain obtained from collecting this information is 3. So if there is no doubt that each agent is willing to get as much information as possible, he may not be willing to provide his / her colleague with the information requested by the latter. This might be the case when the net value of the information exchange is negative for him, i.e. when he provides his colleague with information while not getting anything himself.

Information exchange (when existing) is done according to the following process. First, one agent sends his colleague a *request* for information. Second, the recipient decides whether to answer positively or not. In the first case, we shall say that the agent *commits*.

3.1 Standard Game

Requests and commitments generate four possible attitudes, which may be considered as strategies in a matrix game:

- requests and commits ($R \wedge C$)
- requests and does not commit ($R \wedge \neg C$)
- does not request and commits ($\neg R \wedge C$)
- does not request and does not commit ($\neg R \wedge \neg C$)

On the basis of the previous assumptions the payoff matrix could be then given here under:

Charlene

Rupert		$c_1 = R \wedge C$	$c_2 = R \wedge \neg C$	$c_3 = \neg R \wedge C$	$c_4 = \neg R \wedge \neg C$
	$r_1 = R \wedge C$	(1,1)	(-2,3)	(3,-2)	(0,0)
	$r_2 = R \wedge \neg C$	(3,-2)	(0,0)	(3,-2)	(0,0)
	$r_3 = \neg R \wedge C$	(-2,3)	(-2,3)	(0,0)	(0,0)
	$r_4 = \neg R \wedge \neg C$	(0,0)	(0,0)	(0,0)	(0,0)

If Rupert and Charlene are to make decisions about providing or requesting information, they can act rationally through the standard quantitative game approach, using the above payoff matrix as the matrix of a Nash game.

In this game, strategies r_3, r_4 and strategies c_3, c_4 are strictly dominated, which implies that the game has the same solution set than the 2x2 game comprised of the first two strategies of Rupert, and the first two strategies of Charlene.

The reduced game is nothing but a Prisoner's Dilemma, with (r_2,c_2) as unique Nash equilibrium. The conclusion is obvious : each agent wants the other to provide him / her with information, but simultaneously refuses to commit to the other's request : a typical non –cooperative attitude.

3.2 Game of Deterrence

Let us now consider a variation of the game here above, in which Rupert considers a negative outcome unacceptable, while Charlene considers a non-strictly positive outcome unacceptable.

In other words, the states of the world that a player considers unacceptable are the following:

For Rupert, it is unacceptable to simultaneously request and commit, when Charlene requests as well, but does not commit

For Charlene, it is unacceptable not to request, and to request when Rupert does not commit

Let us assume that in this game, the only thing about which both players really care, is to get an acceptable outcome.

The game matrix is then:

Charlene

Rupert		$c_1 = R \wedge C$	$c_2 = R \wedge \neg C$	$C_3 = \neg R \wedge C$	$c_4 = \neg R \wedge \neg C$
	$r_1 = R \wedge C$	(1,1)	(0,1)	(1,0)	(1,0)
	$r_2 = R \wedge \neg C$	(1,0)	(1,0)	(1,0)	(1,0)
	$r_3 = \neg R \wedge C$	(0,1)	(0,1)	(1,0)	(1,0)
	$r_4 = \neg R \wedge \neg C$	(1,0)	(1,0)	(1,0)	(1,0)

Let us first consider the matrix here above as the matrix of a standard game. It can be easily seen that there are 9 Nash equilibria in the game, i.e. (r_1,c_1) plus the 8 strategic pairs corresponding to the selection by Rupert of either r_2, or r_4

Now let's turn to the game of deterrence. Strategies r_2 and r_4 are safe, implying that c_1, c_2, $c_{3,}$, and c_4 are not positively playable.

It follows that all four strategies of Charlene are playable by default, and consequently that c_1 is deterrent vis-à-vis r_3, and r_2 is deterrent vis-à-vis r_1 and r_3.

On the whole, the game of deterrence has one solution:

$\{J(r_1) = 0; J(r_2) = 1; J(r_3) = 0; J(r_4) = 1; J(c_1) = 0; J(c_2) = 0; J(c_3) = 0; J(c_4) = 0\}$, leading to 8 of the 9 equilibria found in the standard game.

This result derives straightforwardly from the fact that the game is of type E.

Moreover, we see that treating the binary game as a game of deterrence leads to a reduction of the size of the equilibria set. This is just an example of a more general result according to which every positively playable equilibrium of a game of deterrence is a Nash equilibrium in the corresponding standard game, while the reverse is not true.

3.3 Dynamic Games of Deterrence

As environment evolves, so do deterrence systems. Since deterrence places a stress on acceptability thresholds, it seems not unreasonable to assume that changes in deterrence systems are discontinuous.

Hence, the dynamics of a deterrence system can be modeled through a multi-stage game, such that at each stage, the players play a static game of deterrence called a *play*. The decisions in a play are called *controls*. The pair comprised of a stage and a

play defines a *state of the game*. Evolution of the state over time defines a *trajectory*. The game *target* is a subspace of the states space, such that when it is reached, the game is over.

A *strategy* is a function that associates a control with each state of the game.

With respect to outcomes, two particular cases will be considered, which - by analogy with the terminology in differential games - we shall call games with *terminal outcomes (GTO's)*, and games with *integral outcomes (GIO's)*. In GTO's, only outcomes at the target will be taken into account, while in GIO's, a player's outcome in the game will be defined by the product of the outcomes associated with each play of the trajectory followed.

Strategic sets being much larger than control sets, determination of strategies playability properties in a multi-stage game of deterrence, can be can be made easier by establishing a connection between these properties and those of controls.

Thus, it is shown [8] that : given a GTO, a strategic pair s, $\prod(x_1,s)$ a trajectory generated by s from initial state x_1, x_f the terminal point of $\prod(x_1,s)$, $s^i \in s$, and $u^i_f = s^i(x_f)$, then the playability properties of s^i and u^i_f are identical.

Controls playability properties can also be considered not within the plays where these controls are implemented, but in the GTO. It will then be assumed as here above, that playability by default occurs in the absence of positive playability, and playability occurs either through positive playability or through playability by default.

If $x_k \in \prod(x_1,s)$, we shall say that u^i_k is a *safe* (resp. positively playable) *control of player i in the GTO*, if $\exists s^i \in S^i$ such that $s^i_k(x_k) = u^i_k$ and s^i is safe (resp. positively playable).

It can then be shown (ibid) that : if s^i is safe (resp. positively playable, or playable by default), then $u^i_k = s^i(x_k)$ is safe, (resp. positively playable or playable by default) in the GTO.

Similarly, it is shown (ibid) that in a GIO, given a strategic pair s, $\prod(x_1,s)$ a trajectory generated by s from initial state x_1, and $s^i \in s$:

1) s^i is safe (resp. positively playable) iff $\forall x_k \in \prod(x_1,s)$, the control $u^i_k = s^i(x_k)$ is safe (resp. positively playable);

2) s^i is playable by default iff $\exists x_k \in \prod(x_1,s)$ such that $u^i_k = s^i(x_k)$ is playable by default.

4 Example 2

4.1 General Statement and Standard Game

Let us assume that Charlene holds a large amount of information which might be of some interest for other agents of the company. On the one hand, Charlene does not

want to be bothered, since she considers that giving Rupert what he wants, might push Rose, an other colleague, to also ask information. On the other hand, turning a request down, means entering in a conflict with possibly negative consequences.

On their side, Rupert and Rose want to get information, but neither wants a conflict with Charlene.

So, the interaction process is clearly a two stage-one. Charlene plays with Rupert in the first stage, and with Rose in the second one.

Let us assume that the interaction between Charlene and Rupert is represented by the following matrix game :

Charlene

Rupert		**C**	**$\underline{C}$**
	R	((1,1)	((-1,-1)
	$\underline{R}$	((0,2)	((0,2)

where as usual:

C stands for commits (in the present case, gives information if requested)
$\underline{C}$: doesn't commit
R : requests (in the present case, asks for information)
$\underline{R}$: doesn't request

This game is nothing more than the classical Entry Deterrence problem, with two Nash equilibria (R,C) and ($\underline{R}$,$\underline{C}$), the latter being the only sub-game perfect equilibrium.

4.2 Game of Deterrence

256 different games of deterrence can theoretically be derived from the above matrix, depending on the players thresholds. But this number can be dramatically reduced through some reasonable assumptions on the outcomes of the standard game :

1) if an outcome of the standard game is considered acceptable by a player, then any superior one should also be considered acceptable;

2) if an outcome of the standard game is considered unacceptable by a player, then any inferior outcome should also be considered unacceptable;

3) According to the outcomes in the standard game, the players preferences over the set of strategic pairs in the game of deterrence should satisfy the following relations :

- for Rupert : $(R,C) \geq (\underline{R},\underline{C}) = (\underline{R},C) \geq (R,\underline{C})$;
- for Charlene : $(\underline{R},C) = (\underline{R},\underline{C}) \geq (R,C) \geq (R,\underline{C})$;

where "=" and "≥" indicate the indifference and preference relations respectively.

One can easily derive from the above set of conditions that only the 16 following cases remain

	C	C
R	(1,1)	(1,1)
C	(1,1)	(1,1)

case 1

	C	C
R	(1,1)	(1,0)
C	(1,1)	(1,1)

case 2

	C	C
R	(1,0)	(1,0)
C	(1,1)	(1,1)

case 3

	C	C
R	(1,0)	(1,0)
C	(1,0)	(1,0)

case 4

	C	C
R	(1,1)	(0,1)
C	(1,1)	(1,1)

case 5

	C	C
R	(1,1)	(0,0)
C	(1,1)	(1,1)

case 6

	C	C
R	(1,0)	(0,0)
C	(1,1)	(1,1)

case 7

	C	C
R	(1,0)	(0,0)
C	(1,0)	(1,0)

case 8

	C	C
R	(0,1)	(0,1)
C	(1,1)	(1,1)

case 9

	C	C
R	(0,1)	(0,0)
C	(1,1)	(1,1)

case 10

	C	C
R	(0,0)	(0,0)
C	(1,1)	(1,1)

case 11

	C	C
R	(0,0)	(0,0)
C	(1,0)	(1,0)

case 12

	C	C
R	(0,1)	(0,1)
C	(0,1)	(0,1)

case 13

	C	C
R	(0,1)	(0,0)
C	(0,1)	(0,1)

case 14

	C	C
R	(0,0)	(0,0)
C	(0,1)	(0,1)

case 15

	C	C
R	(0,0)	(0,0)
C	(0,0)	(0,0)

case 16

Let us call *conflict* a situation where an request from Rupert is turned down by Charlene

If we want to introduce dynamics in the problem, assumptions must be made about the transition rules from one stage to the next.

Let us consider that at each stage the matrix depends on the outcomes of the previous plays. For instance, it seems reasonable to consider that Charlene will lose more at each conflict (her reputation as a collaborative person might deteriorate), and that if in the beginning the losses may be painless, a protracted series of conflicts will eventually lead to an unacceptable outcome for her.

For the sake of simplicity, we shall assume that:

- one conflict is all what Charlene can stand
- the first play is modeled by case 5 here above
- the second play is modeled by case 6 if there is a conflict in the first play, and by case 5 if not.

Charlene faces a dilemma : if in the one shot game corresponding to case 5, C is safe and deters Rupert to request, this is not true when case 5 is considered to be the first play in a two stage game. Indeed, if a request from Rupert is turned down by Charlene, there is conflict, and hence the second play being modeled by case 6, will be characterized by two possible and quite opposite solutions (in terms of playability).

So Rupert might anticipate that Charlene will consider it hazardous to start a conflict in the first stage. He may then deduce that, since Charlene wants to get an acceptable outcome in the second play, she will avoid a conflict in the first one. Then Rupert will request, and Charlene will give him the information requested. But this is

not sure, since precisely, the result being undetermined in the second play, one cannot exclude that Charlene deters Rose to request in the second play.

Furthermore, were it is not so, Rose may still choose not to request, which guarantees her an acceptable outcome. On the whole, although the game resembles Selten's Chain Store Paradox, there are here solutions for which deterrence works.

5 Congestion Control Analysis

We shall here recall [10] how games of deterrence can model task flow networks, and provide a congestion control. We shall then apply the results already obtained to further analyze how the game type impacts on the risk of congestion in the dynamic workflow problem.

5.1 State Representation

At a given time t, each agent may send a request and/or a commitment throughout the conversational network. Hence, for any pair of agents (i j), there are 4 possible states of i with respect to j . Moreover, requests may be originated either by agents inside or outside the network.

We can then associate with every ordered pair of agents (i,j), a dibit $(A_{ij}(t),B_{ij}(t))$, where $A_{ij}(t)$ and $B_{ij}(t)$ validate (value 1) or inhibit (value 0) *requests* from i to j, and *commitments* by j to a request sent by i, respectively. Thus at each tick of the clock, the state of the agents network can be represented by a square binary matrix.

This state matrix can be considered as the matrix of a game of deterrence with two abstract players, the request function (R) and the commitment function (C) and agent sets as strategic sets. Each row i (resp. each column j) represents a strategy r_i of R, (resp. strategy c_j of C) corresponding to emission of a request by agent i (resp. emission of a commitment by agent j). $(A_{ij}(t),B_{ij}(t))$ is the outcome vector associated with the selection of strategic pair (r_i,c_j) at time t.

A strategy r_i is:

- safe, if requests from i toward all other agents are validated;
- dangerous, if there is at least one agent j toward which request is not validated.

We define similarly safe and dangerous commitment strategies c_j.

Concepts of positive playability, playability by default, playability, and deterrence are then directly applicable. Assume for instance that request r_i is deterrent vis-à-vis commitment c_j.

The usual conditions can be interpreted here as follows:

1) agent i can emit a request;
2) agent j cannot commit to a request sent by agent i;
3) there is another agent j' who can commit to the request sent by agent i.

The game can be replayed at each tick of the clock. So network control can be done through a multi-stage game.

5.2 Congestion and Congestion Control

In the sequel, we shall look for satisficing conditions for organizational efficiency. These conditions can be expressed under the form of a minimal value of task flow, that is under the form of a threshold condition.

Now it is well known in communication networks, that when the input traffic increases beyond a threshold, the information flow across the network slows down, eventually leading to congestion, that is, to a steady state of the network, such that no piece of information can be transmitted across the network.

The usual method for preventing congestion is based on a limitation of the volume of data allowed to enter the network. In order not to let the information flow drop too much, congestion control can be triggered at a threshold level still far from congestion. In other words, the level for which congestion control is triggered, may be parameterized.

In the multi-agent task flow problem, congestion can be defined as a sequence of two consecutive identical states such that no agent is able to request or to commit, which implies that at least one of the two players should have no positively playable strategy

Congestion control is based on the following algorithm:

if the game has positively playable equilibria, one of them is selected and application of transition rules leads to the next state matrix

if there is no positively playable equilibrium, an equilibrium is selected while the network entries are closed, and application of transition rules leads to the next state matrix.

It has been shown that applying this algorithm ensures prevention of congestion.

The state of congestion can be declared straightforwardly from the resolution of the game. But, as already discussed, a game of deterrence might have several solutions, for instance, one where one player has some positively playable strategy, and another where its strategies are playable by default. Should we then rely on the first solution and consider there is no congestion, or should we consider the second solution and "declare" the state of congestion ?

Therefore, to solve he congestion problem, we have to refine the conditions concerning matrix configurations associated with communication network hypothesis, and the type of solutions multiplicity.

5.3 Congestion Avoidance and Graphs of Deterrence

As already mentioned, depending on the graph of deterrence, 7 types of games can be distinguished: type R, type C, type β, type R-C, type R-β, type C-β, type R-C-β.

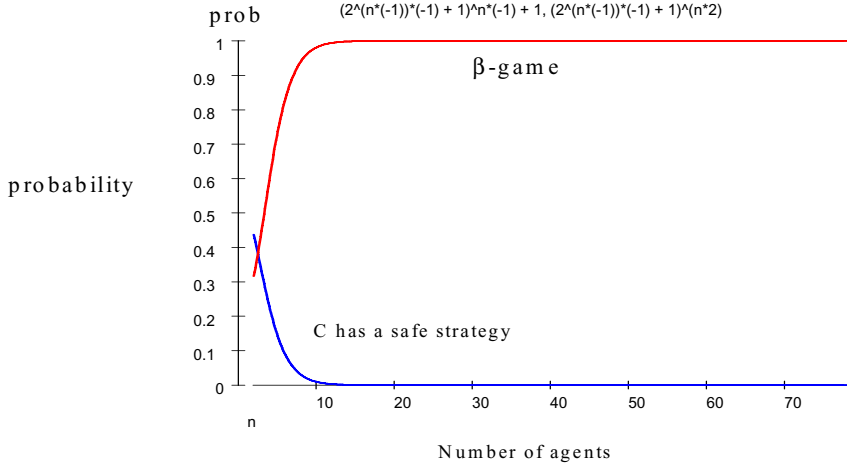
It has been shown [11] that:

(i) there is no congestion for types C,R-C,C-β, R-C-β.
(ii) types β and type R-β can lead to congestion in some cases.
(iii) in type R congestion is always possible.

In order to roughly assess the frequency of occurrence of a particular game type among the seven possible ones, let us first consider a completely flat organization, where we furthermore assume that given any pair (i,j) of agents, i and j may interact as they wish.

The figure hereunder gives the probability of occurrence of a β-type game, and of a C-type game, as functions of the number of agents. It shows that the probability of a β-type game increases and converges very quickly toward 1, while the probability of a C-type game decreases and converges very quickly toward 0 . Roughly, occurrence of a β-type game is almost sure for organizations comprised of more than 10 agents.

So, congestion avoidance cannot be guaranteed in a completely autonomous agent organization as soon as the latter is comprised of more than 10 agents. In other words, small is beautiful!



For large organizations, global self-control cannot be achieved satisfactorily.

One can slice these large organizations into smaller ones – we shall call them *regional*, such that inside each one of these, cooperation works nicely (i.e. the probability of congestion is low). Regional organizations can then be considered as super-agents interacting inside either the global organization itself, if the number of the former is well below 10, or inside some intermediate level organizations, if not. The intermediate organizations can in turn be considered as super-agents inside a higher level organization and so on, till the global organization level is reached. This approach, which for example, is intensively used in the car making industry, is nothing more than a multi-level organization design scheme, suited to provide "smooth" operation from the workflow point of view.

Management intervention occurs only during the organization design stage.

Another possibility consists in keeping the organization as it is (i.e. not slicing it) and using the congestion control algorithm when needed. Here, management intervention does not occur only during the design stage but as soon as the risk of congestion is present.

The curves here above show that the intervention frequency *might* be high as soon as the number of agents is over 10.

Now, the "slicing" technique amounts to nothing more than forbidding an agent to send a request to all other agents or to commit to the requests of all other agents. It follows that in the corresponding state matrix, safe strategies will never show, and therefore games of type R, C, R-C, R, R-β, C-β, and R-C-β.will never show. So we come back to the β-type game case.

As already pointed, in such a game congestion *might* occur, but it is not inescapable. Here different sub cases can be considered, depending on whether the graph is comprised of:

- simple loops or circuits
- general graphs with circuits.

In the first case, it is easy to see that there is only one solution for which all strategies are playable by default, and hence congestion occurs.

The second case is more tricky, as we can have circuits inside circuits which induce multiple solutions and here we must define the "right transition". It is then necessary to analyze exhaustively all possible graph structures.

6 Example 3

Let us consider a two agent team, in which we shall assume for the sake of simplicity that tasks take only one time unit to be accomplished.

Let us assume moreover that transitions are *only* governed by the two following rules which state that an agent is not allowed to:

- send two consecutive requests to the same agent (be it himself)
- commit consecutively twice to the request of the same agent (be it himself)

Since, unlike example 2, the game of deterrence is not derived from a standard game, we consider all 256 matrices are possible

Moreover, let us assume that the game starts at time t_0, with play P_0 characterized by the following matrix:

C

R	$\mathbf{c_1}$	$\mathbf{c_2}$
$\mathbf{r_1}$	(1,1)	(1,0)
$\mathbf{r_2}$	(0,1)	(1,0)

The game exhibits one positively playable equilibrium (r_1,c_1) (the game type is R-C). Hence here there is non congestion

At the next stage, the game matrix is (play P_1):

C

R	$\mathbf{c_1}$	$\mathbf{c_2}$
$\mathbf{r_1}$	(0,0)	(1,0)
$\mathbf{r_2}$	(0,1)	(1,0)

It can be easily shown that the game is a β-type one, with all strategies being playable by default.

This means in turn that the congestion control algorithm has to be triggered. Since (r_1,c_1) has been the last strategic pair selected, transition rules forbid it to be selected at this stage. Consequently, the game exhibits three *feasible* equilibria: (r_1,c_2), (r_2,c_1) and (r_2,c_2).

Let us first consider the play P_2 obtained by selection of strategic pair (r_1,c_2):

C

R		$\mathbf{c_1}$	$\mathbf{c_2}$
	$\mathbf{r_1}$	(1,1)	(1,0)
	$\mathbf{r_2}$	(0,0)	(1,0)

It be can be shown that the game has two solutions:

- $\{J(r_1) = 1 ; J(r_2) = 0 ; J(c_1) = 0 ; J(c_2) = 0\}$
- $\{J(r_1) = 1 ; J(r_2) = 0 ; J(c_1) = 1 ; J(c_2) = 0\}$

In the first solution player C's strategies are playable by default, while in the second solution, c_1 is positively playable. We know that then strategic pair (r_1,c_1) may be selected in which case no congestion needs to be declared.

The transition rules then take the game back to play P_1.

Let us next consider the play P_3 obtained by selection of strategic pair (r_1,c_2). The game matrix is:

C

R		$\mathbf{c_1}$	$\mathbf{c_2}$
	$\mathbf{r_1}$	(1,1)	(0,0)
	$\mathbf{r_2}$	(0,1)	(1,0)

It be can be shown that the game has two solutions:

- $\{J(r_1) = 0 ; J(r_2) = 0 ; J(c_1) = 1 ; J(c_2) = 0\}$
- $\{J(r_1) = 1 ; J(r_2) = 0 ; J(c_1) = 1 ; J(c_2) = 0\}$

As in the previous case, strategic pair (r_1,c_1) may be selected without declaring congestion, and the game is back to play P_1

Last, let us consider the play P_4 obtained from play P_1 by selection of strategic pair (r_2,c_2) :

C

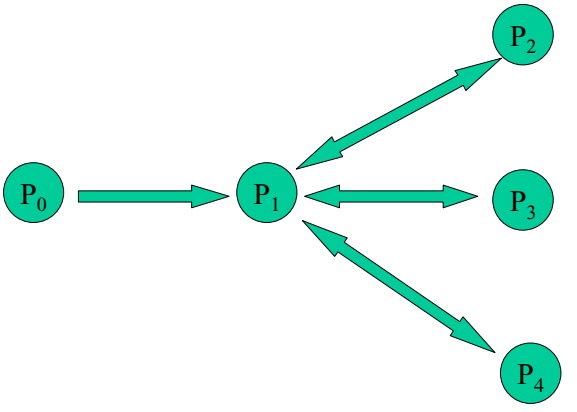
R		$\mathbf{c_1}$	$\mathbf{c_2}$
	$\mathbf{r_1}$	(1,1)	(1,0)
	$\mathbf{r_2}$	(0,1)	(0,0)

The game has only one solution:

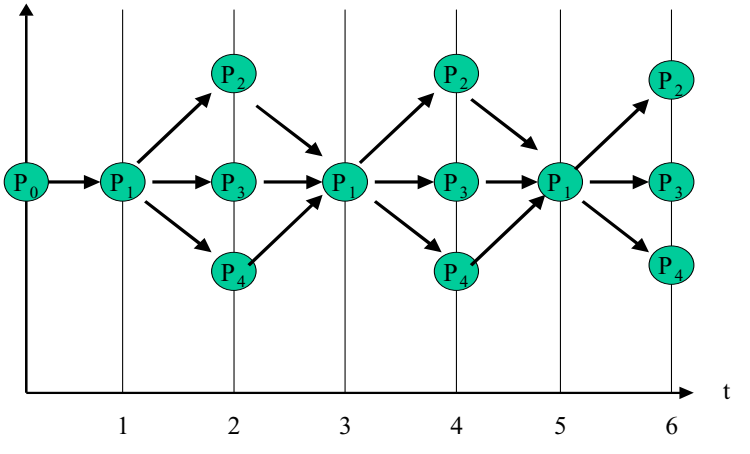
$\{J(r_1) = 1 ; J(r_2) = 0 ; J(c_1) = 1 ; J(c_2) = 0\}$,

which means that once again strategic pair (r_1,c_1) is selected, with transition rules taking the game back to play P_1.

It follows that whatever the duration of the game, the only plays that can be encountered are P_0, P_1,P_2,P_3 and P_4, with implications and game trajectories as given by the figures hereunder:



Plays interactions



Game trajectories

On the whole, since play P_1 is the only play associated with the state of congestion, whatever the transition selected when the game is at play P_1, there will be no congestion at the next stage of the game, while on the opposite, whatever the play at an even stage, congestion should be declared at the next stage.

References

1. Bestougeff, H.: Self regulating teams and games of deterrence, Proceedings 2nd World Multiconference on Systemics, Cybernetics and Informatics, Orlando 1998
2. Bestougeff, H., Bouaissi, A.: Multi-Agent Architecture to model and simulate a business network of recurrent conversations and associated tasks, Proceedings EMCSR-98-From Agent Theory to Agent Implementation, Vienne, 1998
3. Bestougeff, H., Bouaissi, A.: Dynamic BPR using a Task/ Communication Model, ACM SIGGROUP Bulletin Vol 18 N°3 1997
4. Bestougeff, H., Rudnianski, M.: Games of deterrence and satisficing models applied to business process modeling, Proceedings AAAI Spring Symposium, Stanford 1998
5. Menasché, D.S, Figuereido, D.R, De Souza e Silva, E.: An Evolutionary Game Theoretic Approach to Congestion Control, Performance Evaluation 62, pp 295-312, Elsevier 2005
6. Rudnianski, M.: Deterrence Typology and Nuclear Stability. A Game Theoretical Approach, in Defense Decision Making, R. Avenhaus, H. Karkar, M. Rudnianski, editors, pp 137-168, Springer Verlag, Heidelberg, (1991)
7. Rudnianski, M.: Deterrence, Fuzzyness and Causality, Proceedings ISAS 97, World Multiconference on Systemics, Cybernetics and Informatics, Caracas 1997
8. Rudnianski, M. Multipolar Deterrence in a Dynamic Environment, IEEE Systems, Man and Cybernetics 1995, vol 5, pp 4279 – 4285.
9. Rudnianski, M., d'Assignies A.: From MAD to MAD, Strategic Stability in the Post-Cold War World and the Future of Nuclear Disarmament, M.L. Best, J. Hughes-Wilson, A. Piontkowsky, editors, NATO ASI series, Disarmament Technologies, vol 3, pp 229-259, Kluwer Academic Publishers, 1995
10. Rudnianski, M., Bestougeff, H.: Modeling Task and Teams through Game Theoretical Agents, Formal Approaches to Agent Based Systems 2000, Lecture Notes in Artificial Intelligence, 1871, pp235-249, Springer Verlag 2001
11. Rudnianski, M., Bestougeff, H.: Modeling Multi-agent Interactions Through State Transitions in Games of Deterrence , Proceedings AAAI Spring Symposium, Stanford 2001
12. Rudnianski, M. , Bestougeff, H.: Modeling Traffic Control Through Deterrent Agents, Formal Approaches to Agent based Systems 2002, pp286-289, Lecture Notes in Computer Science 2699, Springer Verlag, 2003

Radical Concepts for Self-managing Ubiquitous and Pervasive Computing Environments

Roy Sterritt[1] and Mike Hinchey[2]

[1] University of Ulster
School of Computing and Mathematics, Jordanstown Campus, BT37 0QB
Northern Ireland
r.sterritt@ulster.ac.uk
[2] NASA Goddard Space Flight Center
Software Engineering Laboratory
Greenbelt, MD 20771
USA
michael.g.hinchey@nasa.gov

Abstract. Autonomous and Autonomic Systems (AAS) are essentially concerned with creating self-directed and self-managing systems based on biologically-inspired metaphors such as the mammalian autonomic nervous system. Future Ubiquitous and Pervasive computing environments will depend on such a self-managing infrastructure. Agent technologies have been identified as a key enabler for engineering autonomy and autonomicity in systems, both in terms of retrofitting self-management into legacy systems and designing and developing totally new systems. Handing over responsibility to the systems themselves raises many concerns for humans. This paper reports on the continued investigation into a strand of research on how to engineer self-protection mechanisms into systems to assist in providing confidence regarding the appropriateness of systems utilizing principles of autonomy and autonomicity. This includes utilizing the apoptosis metaphor to potentially provide a self-destruct signal between autonomic agents as and when needed, and an ALice signal to facilitate self-identification and self-certification between anonymous autonomous agents and systems.

1 Introduction

The driving force behind future paradigms of computing is the increasing convergence between the following technologies: (1) proliferation of devices; (2) wireless networking; and (3) mobile software [1].

Weiser first described what has become know as Ubiquitous Computing [2] as the move away from the "dramatic" machine (where the focus of hardware and software was on being so exciting that we, as users, would not want to be without it) towards making the machine "invisible" (being so embedded in our lives that it is used without thinking of it, nor recognizing it as computing per se). Behind Ubiquitous Computing and related terms and research areas, lie three key properties of computing [1]: being (1) nomadic; (2) embedded; and (3) invisible.

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 370 – 378, 2006.

In effect, this may lead to the creation of a single system with (potentially) trillions of *networked information devices*. All of these next generation paradigms, in one form or another, will require an autonomic–self-managing–infrastructure to enable the successful achievement of this envisaged level of invisibility and mobility.

Pervasive Computing is distinct from Ubiquitous Computing in that the computing devices are not invisible, but exist all around us. Increasingly, users are accessing computing resources via personal devices. Personal devices offer unique challenges for self-management due to their multi-equipment, multi-situation, and multi-user nature. Personal or Pervasive Autonomic Computing is much less about achieving optimum performance or exploiting redundancy (as in Autonomic Computing itself) and more about simplifying use of the equipment and the associated services [3],[4]. As such, it is particularly relevant to dealing with the move towards future nomadic, embedded and invisible computing [5],[6],[7].

2 Biologically-Inspired Computing Self-managing Concepts

An Autonomic System aims to achieve self-management through *self* properties such as self-configuration, self-healing, self-optimization, self-protection via self-awareness, self-situation, self-monitoring and self-adjusting (for more information, see [8]).

The autonomic environment requires that autonomic elements and, in particular, autonomic managers communicate with one another concerning self-* activities, in order to ensure the robustness of the environment. Fig. 1 depicts that the autonomic manager communications (AM⇔AM) also includes a reflex signal. This may be facilitated through the additional concept of a pulse monitor—PBM (an extension of the embedded system's HBM, or heart-beat monitor, which safeguards vital processes

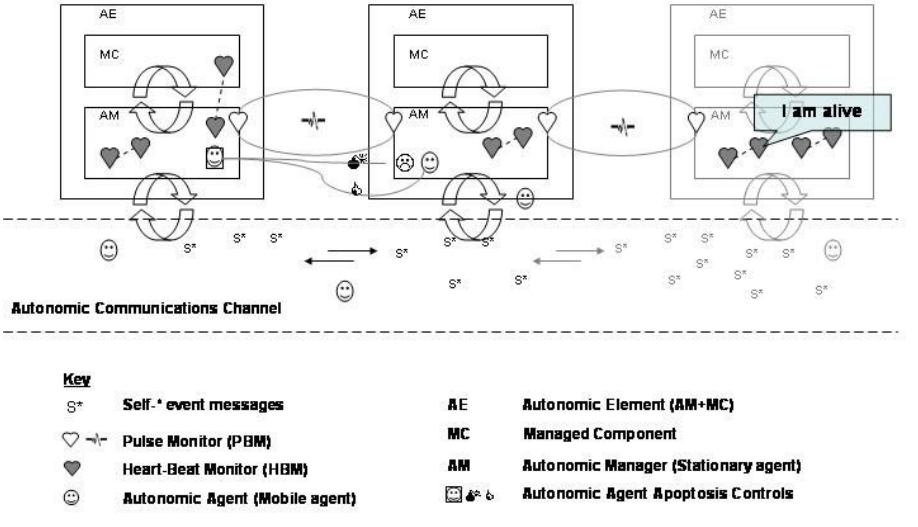


Fig. 1. Autonomic and Autonomous Computing Environment (AACE)

through the emission of a regular "I am alive" signal to another process) with the capability to encode health and urgency signals as a pulse [9]. Together with the standard event messages on the autonomic communications channel, this provides dynamics within autonomic responses and multiple loops of control, such as reflex reactions among the autonomic managers [10].

2.1 Autonomic Reflex Signal – Lub-Dub Pulse Emission

This reflex component may be used to safe-guard the autonomic element by communicating its health to another AE [6]. The component may also be utilized to communicate environmental health information [3, 7]. For instance, in the situation where each PC in a LAN is equipped with an autonomic manager, rather than each of the individual PCs monitoring the same environment, a few PCs (likely the least busy machines) may take on this role and alert the others through a change in pulse to indicate changing circumstances.

An important aspect concerning the reflex reaction and the pulse monitor is the minimization of data sent—essentially only a "signal" is transmitted. Strictly speaking, this is not mandatory; more information may be sent, bandwidth permitting, yet the additional information must not compromise the reflex reaction. For instance, in the absence of bandwidth concerns, information that can be acted upon quickly and not incur processing delays could be sent. The important aspect is that the information must be in a form that can be acted upon immediately and not involve processing delays (such as is the case of event correlation).

Just as the beat of the heart has a double beat ("lub-dub", as it is referred to by physicians) the autonomic element's pulse monitor (Fig. 2) may have a double beat encoded—as described above, a *self* health/urgency measure and an *environment*

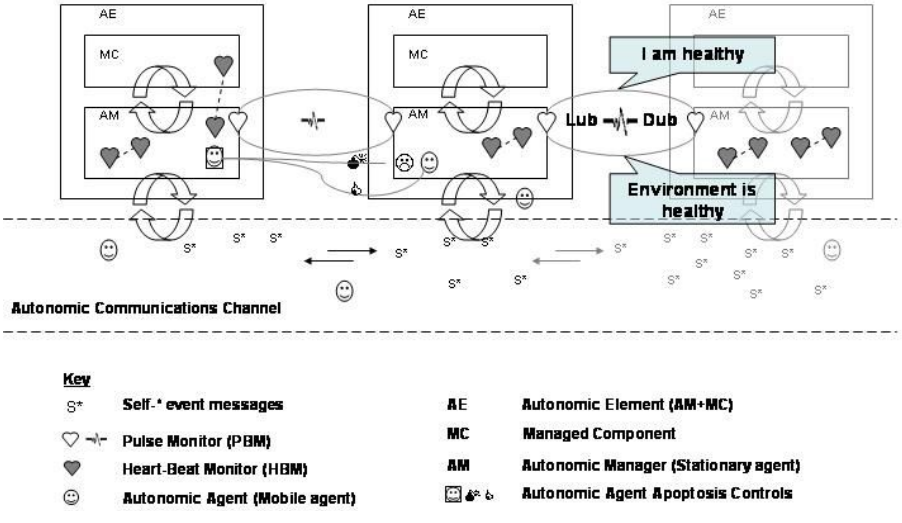


Fig. 2. AACE incorporating the Lub-Dub Pulse Emission and Monitoring

health/urgency measure. These match directly with the two control loops within the AE, and the self-awareness and environment awareness properties.

2.2 The ALice Signal

An aspect to this research is that Anonymous Autonomous/Autonomic Agents (and indeed signals and communications—event messages, etc.) need to work within the Autonomic System to facilitate self-management. As such, the agents and their hosts need to be able to identify each other's credentials through such means as an *ALice* (***A***utonomic ***Lice***nse) signal (Fig. 3). This would allow a set of communications to ensure that the visiting mobile agent (or signal/communications) has valid and justified reasons for being there, as well as providing security mechanisms for the visiting agent during interaction with other agents and host. An unsatisfactory ALice exchange may lead to self-destruction of that agent as a means of self-protection for the system.

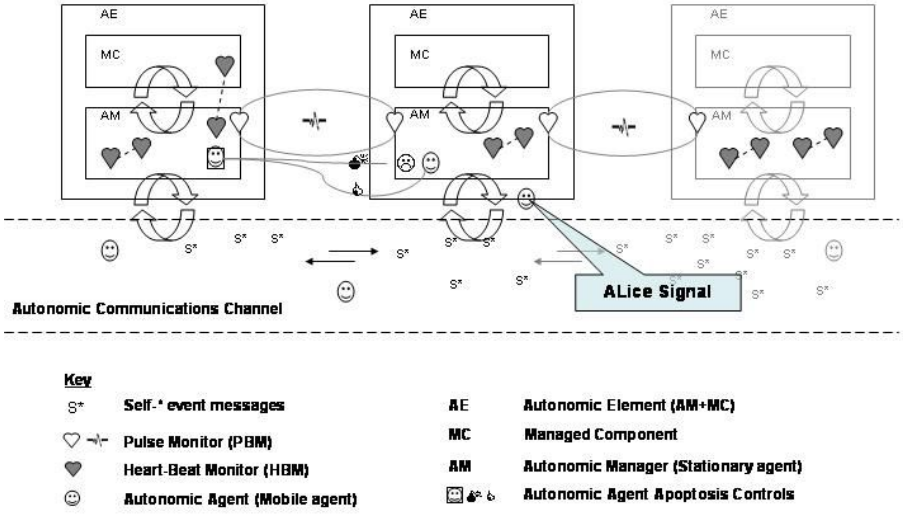


Fig. 3. AACE incorporating the ALice Signal

3 Pervasive Exemplar: Smoke Alarms

Smoke Alarms offer a simple pervasive system example that can take advantage of some of the Autonomic Concepts described above in innovative ways. Although Autonomicity is specifically motivated for addressing complex systems, the point to remember is that we are moving into an era with the potential of having trillions of pervasive devices.

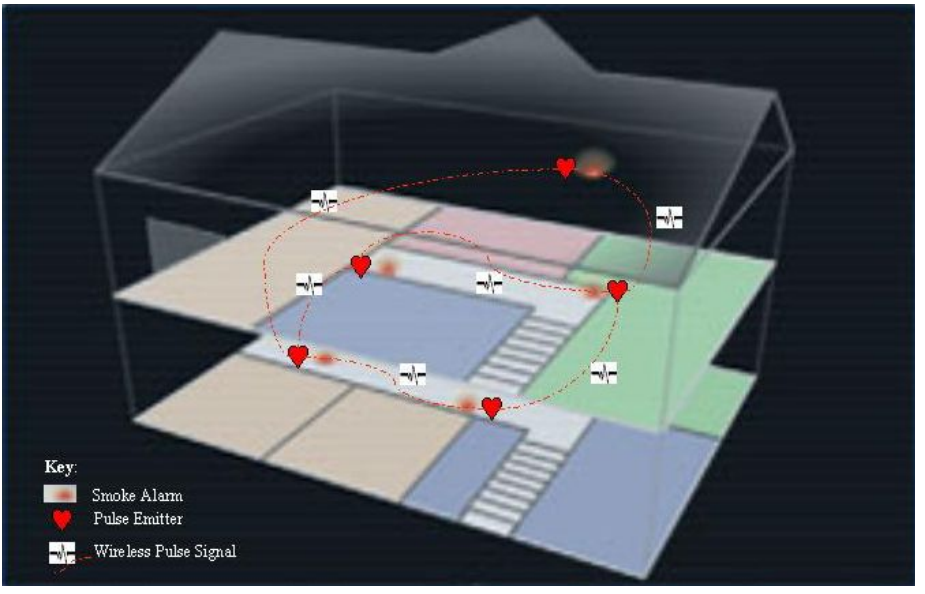


Fig. 4. Making Homes "Smarter": Autonomic Smoke Alarms

3.1 A "Smart" Smoke Alarm

Many jurisdictions now require the use of smoke alarms in private homes (and in places of business). Many people use individual alarms, ideally one per level in the building, and which are powered by batteries. It is generally recommended that batteries are replaced on a regular basis, perhaps every year or every two years, depending on the alarm itself and its rate of power consumption.

In newer construction, hard-wired smoke alarms are available, powered by an electricity source, with batteries only used for backup in the event of a loss of power. These hard-wired alarms also have the advantage that when smoke is detected by one alarm, all of the alarms in the building will sound, whereas with the individual alarms, only the alarm that has sensed smoke will be activated.

We know of only one commercially-available smoke alarm system where battery-operated smoke alarms will all activate when one of the alarms senses smoke. The use of principles from autonomic computing, and the biologically-inspired mechanisms described earlier, allow us to develop this further and provide a more innovative, practical, and efficient smoke alarm system, achieving the advantages of a hard-wired system, but the flexibility of a wireless system.

Consider a two-storey house, with a single (unwired) standard smoke alarm on each level. If there is a fire on the lower level, which smoke alarm will sound first will depend on the location of the fire and on the positioning of the alarms. Suppose that the lower level alarm is not working correctly due to a low battery; then, the upper level alarm will only sound when smoke is detected at that level, losing valuable time and most probably endangering lives since smoke kills.

Most smoke detectors will sound intermittently to indicate that the battery is low, but if the alarm is located in an out-of-the-way location, or somewhere where the

emissions are not going to be heard, the low-battery situation may not be detected. Hard-wired alarm systems have an advantage here in that when one alarm has a low battery, all of the other alarms linked to it may also sound.

Let us now consider a wireless autonomic smoke alarm system (as depicted in Figure 4). Each smoke alarm emits a "lub-dub" signal, using the dual beat to indicate firstly that the alarm is functioning correctly (its heartbeat) and secondly its current "health" (functioning correctly, battery level, whether smoke has been detected, etc.). (Note the frequency of the lub-dub is very low, when everything is operating as normal, compared to the standard autonomic element as described elsewhere, so as to minimize battery usage.)

When smoke is detected by a single smoke alarm in the system, this information is propagated to all of the alarms in the system, and all of the alarms in the building can be sounded almost immediately, although only one of them has actually detected smoke.

Should one of the alarms in the system fail (because of a low battery, or for other reasons), the rest of the system will not be affected (as a hard-wired system may be).This is because the rest of the system is not dependent on the non-functioning alarm, which may be ignored because it will be put into self-destruct mode (or self-quiescence mode, which is a less extreme form of self-destruction, which can be reversed [11]) as a result of an unsatisfactory ALice signal exchange, indicating to the rest of the system that that particular device is not functioning correctly.

Thus, using autonomic properties and biological-inspiration based on the techniques we described earlier, we can create a wireless smoke alarm system that offers both the advantages of a wireless system (no wiring required, solely powered by batteries, easy to install in existing construction) and also the advantages of a hard-wired system (one alarm detecting smoke can result in all of the alarms sounding, or one alarm having a low battery can be signalled on all of the alarms in the system). It also overcomes one major failing of many hard-wired systems, namely that an entire system can fail due to the failure of a single alarm.

The approach described here, based on autonomic computing principles, offers the same advantages of both the wireless and wired alarms, but without the disadvantages. It also offers greater robustness in the case of failure by virtue of self-management, and offers the potential to extensively expand the functionality of smoke alarms, for instance together with an appropriate computer-based smart home system, provide adaptive and dynamic fire exit routes through lighted direction signals to indicate the best escape route taking into consideration place of fire, crowds, blocked fire exits, etc. New solutions such as these are increasingly being considered critical due to perceived threat of terrorist attacks. Any such pervasive sensor and effector system will require autonomicity and autonomy-based techniques.

3.2 Implementing the Autonomic Smoke Alarm

This technology exploits our "lub-dub" technology, as first described in [12], using the lub-dub "double heart-beat" to carry "health" information. In this case that information is that all smoke detectors in a building are operating correctly (i.e., if one smoke detector were faulty, or were to lose power, such as battery failure, etc., this would be detected by the failure to receive a heartbeat within a specified period of time), and the

second beat (the "dub") would carry operational data (that is, if an alarm signal needed to be sounded by one device, the second beat would be used to convey this information to the other smoke detectors in the building so that they too would switch to alarm mode).

While we see the smoke alarm system as a particularly useful application of the lub-dub technology, there are many more possible uses: burglar alarm systems, sprinkler systems, satellites, and, in fact, any so-called "sensor network", where a number of simple sensors are used together in a way that collects information regarding the status of various devices. In all of these applications, the first beat of the lub-dub could be used to carry health information (i.e., that the device is functioning correctly and forwarding data readings in a timely fashion) and the second to carry instructions or signals, either trigged by a particular event (e.g., detection of smoke) or to perform various operational functions.

This smoke detector is unique in that it links devices within a building together wirelessly, offering functionality that would normally only be available in a hardwired system. More importantly, it provides a mechanism to determine if there are faulty devices rather than discovering this by failure of the device when most needed. The approach allows for all devices to signal an alarm rather than just the device that detects the smoke, and also to send a signal that is recognizable throughout the building if one of the devices fails (rather than with current devices where a low battery causes a signal, such as beeping, only at the failing device, all devices could signal the problem).

Again, this is a particular instantiation of the approach. It has applications in a large number of areas above and beyond a smoke detector.

This is a particular instantiation of an idea to exploit our lub-dub technology. Lub-dub is significantly more general and can be used in many more application areas and in many more ways. However, this is a specific product that could be adapted to many uses both in NASA missions based on sensor networks (or which there are several with many forthcoming) and in everyday life. Anywhere monitoring is required, we believe this approach could be used.

The Autonomic Smoke Detector is a particularly illustrative example that highlights the technique, but is also something that could likely be developed as a product. Current smoke detectors "beep" when their battery is low. This approach would allow for all detectors in the house/building to indicate such a problem, rather than just on the level that has the problem. Also, it allows for existing construction to have the advantage that the alarm will sound on each level rather than just on the level where the smoke has been detected (as can be achieved with hardwired systems built into new construction). This could potentially save lives.

4 Conclusions

Autonomic and Autonomous Systems techniques provide a means to offer emerging pervasive, ubiquitous and ambient intelligence systems a means to self-manage. In this paper we have taken a simple example, that of smoke alarms, to illustrate the techniques that are emerging in self-managing complex computer-based systems and which may also be of use in the pervasive computing space.

Most jurisdictions require the installation of smoke detectors (and often sprinkler systems) in new housing. For most new houses, smoke detectors are hardwired and connected together so that if one is activated, all of the smoke detectors sound the alarm. For existing houses, installing such wiring is nontrivial. Moreover, a better mechanism would be to have smoke detectors within a house (or building) communicate with each other wirelessly, using the concept of a heartbeat to ensure that each device was working correctly (with appropriate action taken if not). In this way, failure of one detector could be identified, and when one detector were activated, the signal could be sent to all devices, using the second beat of our "lub-dub" technology.

Although, a simple smoke alarm system has been described here for purposes of illustration, clearly, many of these techniques will apply to similar systems such as burglar alarm and access-control systems, wireless sensor networks, and other systems based on sensors and actuators.

Acknowledgements

This research is partly supported at University of Ulster by the Computer Science Research Institute and the Centre for Software Process Technologies (CSPT) which is funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

Part of this work has been supported by the NASA Office of Systems and Mission Assurance (OSMA) through its Software Assurance Research Program (SARP) project, Formal Approaches to Swarm Technologies (FAST), and by NASA Goddard Space Flight Center, Software Engineering Laboratory (Code 581).

Some of the technologies described in this article are patent pending and assigned to the United States government.

References

1. The Future of Computing Project; http://www.thefurtureofcomputing.org, 2004.
2. Weiser, M., "Creating the Invisible Interface," In *Proceedings Symposium on User Interface Software and Technology*, New York, NY, ACM Press, 1994.
3. Bantz, D.F., Bisdikian, C., Challener, D., Karidis, J.P., Mastrianni, S., Mohindra, A., Shea, D.G., and Vanover, M., Autonomic personal computing, *IBM Sys J* 42(1) pp 165-176, 2003
4. Bantz, D.F. and Frank, D., "Challenges in Autonomic Personal Computing, with Some New Results in Automatic Configuration Management," In *Proceedings of IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003)* at INDIN 2003, Banff, Alberta, Canada, 22-23 August 2003, pp451-456.
5. Sterritt, R. and Bantz, D.F., "PAC-MEN: Personal Autonomic Computing Monitoring Environments," In *Proceedings of IEEE DEXA 2004 Workshops - 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 04),* Zaragoza, Spain, August 30 – 3 September, 2003.

6. Sterritt, R. and Chung, S., "Personal Autonomic Computing Self-Healing Tool," In *Proceedings of IEEE Workshop on the Engineering of Autonomic Systems (EASe 2004)* at 11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004), Brno, Czech Republic, 24-27 May, pp 513-520.
7. Sterritt R. and Bantz, D.F, "Personal Autonomic Computing Reflex Reactions and Self-Healing", *IEEE Transactions on Systems, Man and Cybernetics*, Part C, Vol. 36, No. 3, IEEE, ISSN 1094-6977, Pages 304-314, May 2006
8. Sterritt, R., "Autonomic Computing", *Innovations in Systems and Software Engineering: a NASA Journal*, 1(1):79-88, April 2005.
9. Sterritt, R., "Pulse Monitoring: Extending the Health-check for the Autonomic GRID," In *Proceedings of IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003)* at INDIN 2003, Banff, Alberta, Canada, 22-23 August 2003, pp 433-440.
10. Sterritt, R. and Bustard, D.W., "Autonomic Computing: a Means of Achieving Dependability?," In *Proceedings of IEEE International Conference on the Engineering of Computer Based Systems (ECBS'03),* Huntsville, Alabama, USA, April 7-11 2003, pp 247-251.
11. Sterritt, R. and Hinchey, M.G., "Biologically-Inspired Concepts for Autonomic Self-Protection in Multiagent Systems," In *Proceedings Workshop on Safety and Security in Agent-Based Systems (SASEMAS 2006)* at Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, 8-12 May 2006.
12. Sterritt, R., and Hinchey, M.G., "SPAACE:: Self Properties for an Autonomous and Autonomic Computing Environment," In *Proceedings Software Engineering Research and Practice 2005 (SERP 2005)*, 27-30 June 2005, Las Vegas, NV, CSREA Press.

Survivable Security Systems Through Autonomicity

Roy Sterritt[1], Grainne Garrity[1], Edward Hanna[2], and Patricia O'Hagan[2]

[1] University of Ulster
School of Computing and Mathematics, Jordanstown Campus, Northern Ireland
R.Sterritt@ulster.ac.uk,
Grainne@coresystems.biz
[2] Core Systems
Belfast
Northern Ireland
Edward@coresystems.biz,
Patricia@coresystems.biz

Abstract. Technological developments such as biometrics are providing new potential for next generation security systems. At the same time these developments can make the system more complex to manage. Some classes of systems have a fundamental requirement to survive be that to ensure an organization does not loose tens of millions of dollars due to downtime or to ensure there is not a security breach. Autonomic self-managing systems are motivated to hide the ever increasing complexity in today's systems but through their selfware approach they also offer the potential to create survivable systems. This paper details one such approach, to create a survivable security system for correction centers.

1 Introduction

In 2001, IBM presented its view on the state of the IT industry [1]—a massive ever growing complexity and cost burden—resulting in the Autonomic Computing initiative and self-managing systems in general becoming mainstream to address the complexity problem, the total cost of ownership, as well as to provide the way forward to enable future pervasive and ubiquitous computation and communications [2][3][4]. The strategic holistic self-managing vision is dependant on many existing research domains not least autonomous computing and fault-tolerant computing (FTC). Carrying on this FTC aspect, self-management potentially can facilitate survivable, dependable and fault tolerant systems.

To enable self-management (*autonomicity*) a system requires many *self* properties (*self-** or *selfware*), such as self-awareness. This paper first explores autonomy versus autonomicity and reflects upon some existing three tier architectures to establish design guidance. The paper then presents a deployed critical security system that requires built in survivability and autonomicity. This Core architecture for access control to a correction center secure location incorporates such identification technologies as biometrics. The implication of these multiple components is that the system becomes more complex, a theme the Autonomic initiatives aim to address. In this situation the self-healing and self-protections aspects are more critical to the system. The continuous monitoring of the system health through control loops

M.G. Hinchey et al. (Eds.): WRAC 2005, LNAI 3825, pp. 379 – 389, 2006.

matches the architecture needs. Research within the Autonomic field such as self-healing can assist in creating a survivable environment [5-9].

2 Autonomicity and Autonomy

In terms of computer-based systems design paradigms, autonomy implies 'self-governance' and 'self-direction' whereas autonomic implies 'self-management'. That is, Autonomy (self-governance) is the delegation of responsibility to the system to meet the defined goals of the system (automation of responsibility including some decision making for the success of tasks) whereas Autonomicity is the self-management of the system (automation of responsibility including some decision making for the successful operation of the system), and as such will often be in addition to self-governance to meets one's own required goals. It may be argued at the systems level that the success of autonomy requires successful autonomicity i.e. ultimately to ensure success in terms of the tasks requires that the system is reliable. For instance, the goals of a system (potentially implemented as mobile agents) may be to find the best price for a holiday from online travel bureaus, the system may have autonomy (self-governance/self-direction) to negotiate between certain price ranges, destinations and dates. The goals to ensure the system is fault tolerant and continues to operate under fault conditions for instance, would not fall under this specific delegated task of the agents (their autonomy), yet ultimately the task may fail if the system can not cope with dynamic changes and conditions in the environment. From this perspective, the autonomic and self-management initiatives may be considered specialized forms of autonomy, that is the autonomy (self-governance, self-direction) is specifically to manage the system (to heal, protect, configure, optimize and so on).

Table 1. Dictionary definitions [10,11]

au·to·nom·ic (àwtə nómmik)
adj.
1. *Physiology.*
   a. Of, relating to, or controlled by the autonomic nervous system.
   b. Occurring involuntarily; automatic: *an autonomic reflex.*
2. Resulting from internal stimuli; spontaneous.

au·ton·o·mic·i·ty (àwtə nóm i síttee)
n.
1. The state of being autonomic.

au·ton·o·mous (aw tónnəməs)
adj.
1. Not controlled by others or by outside forces; independent: *an autonomous judiciary; an autonomous division of a corporate conglomerate.*
2. Independent in mind or judgment; self-directed.
3.
   a. Independent of the laws of another state or government; self-governing.
   b. Of or relating to a self-governing entity: *an autonomous legislature.*
   c. Self-governing with respect to local or internal affairs: *an autonomous region of a country.*
4. Autonomic.

[From Greek `autonomos : auto-`, *auto-* + `nomos`, *law*]

Taking the dictionary definitions of autonomous and autonomic [10,11], autonomous essentially means 'self-governing' (or 'self-regulating', 'self-directed') as defined. 'Autonomic' is derived from the noun 'autonomy', and one definition of autonomous is autonomic, yet the main difference in terms of dictionary definitions would be in terms of speed; autonomic being classed as 'involuntarily', 'reflex' and 'spontaneous'.

When one considers the tiers for Intelligent Machine Design [12-15] it consists of top level being reflection, middle level routine and bottom level reaction. Reaction is the lowest level where no learning occurs but immediate response to state information coming from sensory systems. Routine is the middle level where largely routine evaluation and planning behaviors take place. It receives input from sensors as well as from the reaction level and reflection level. Reflection, a meta-process where the mind deliberates about itself, top level receives no sensory input or has no motor output, it receives input from below. The levels mark a degree of speed, for instances; reaction should be (as close to) immediate; whereas reflection is consideration over time.

Table 2. Similarity between three tier architectures form various research domains

Intelligent Machine Design	**Future Comms Paradigms**	**DARPA/ISO Autonomic Information Assurance (AIA)**	**NASA's Science Missions**	**Self-Directing & Self-Managing Systems Potential General Architecture**
Reflection	Knowledge Plane	Mission Plane	Science	Autonomous
Routine	Management / Control Plane	Cyber Plane	Mission	Selfware
Reaction	Data Plane	Hardware Plane	Cmd Sequence	Autonomic

Other variations of this three tier architecture have been derived in other research domains (summarized in Table 2):

In the future communications paradigms research domain a new construct, a knowledge plane, has been identified as necessary to act as a pervasive system element within the network to build and maintain high level models of the network. These indicate what the network is supposed to do to provide communication services and advice to other elements in the network [16]. It will also work in coordination with the management plane and data planes. This addition of a knowledge plane to the existing data & management/control places would form a three tier architecture: data—management/control—knowledge [16].

In the late 1990s DARPA/ISO's Autonomic Information Assurance (AIA) program studied defense mechanisms for information systems against malicious adversaries. The program specified an architecture consisting of three plans: mission, cyber and hardware with one finding from the research being that fast responses are necessary to counter advance cyber-adversaries [17].

NASA's science mission management, from a high level perspective, may be classified into Science Planning: setting the science priorities and goals for the mission; Mission Planning: involving the conversion of science objectives to instruments and craft maneuvering, house keeping and scheduling, comms link etc; to Sequence Planning: production of detailed command sequence plans. Overall the transformation from high level goals to implementation on the hardware [18].

These versions of a high level three tier view of self-governing and self-managing systems may be generalized into autonomous—self-ware—autonomic tiers. Of course this is not intended to be prescriptive nor conflict with other views of autonomic systems etc, the intention to examine and view systems in this light is to assist in developing effective systems.

'Autonomic' became mainstream within Computing in 2001 when IBM launched their perspective on the state of information technology [1]. Autonomic Computing has four key self properties: *self-configuring*, *self-healing*, *self-optimizing* and *self-protecting* [19]. In the few years since, the *self-x* list has grown as research expands, bringing about the general term *selfware* or *self-**, yet these four initial self-managing properties along with the four enabling properties; *self-aware (of internal capabilities and state of the managed component)*, *self-situated (environment and context awareness)*, *self-monitor* and *self-adjust (through sensors, effectors and control loops)*, cover the general goal of self management [20].

3 The Core Architecture

Locations, campuses and sites that require high security access, such as research labs and law enforcement correction centers, are increasingly seeking to utilize the emerging technology in biometrics such as iris and finger print technologies to allow and restrict access and movement around their locations. Secure locations have additional architectural requirements to the norm for dependable systems, such as the architecture should be secure, fault-tolerant, and non-exploitable [17]. That is, the system must meet the security policies of the organization and it must accommodate different security infrastructures; the system must remain robust and secure when faults occur, both random faults caused by the failure of system elements and to malicious faults caused by a deliberate attack on the system; that communication must be reliable so that defensive components remain fully functional even in the face of an attack on the infrastructure; and that it must not be possible for an attacker to exploit defensive components to effect an undesirable action, for example, an attacker or random fault must not be able to trigger a response that causes the system to unnecessarily deny legitimate users access and movement nor the opposite allow non-legitimate users access and movement around the location [17].

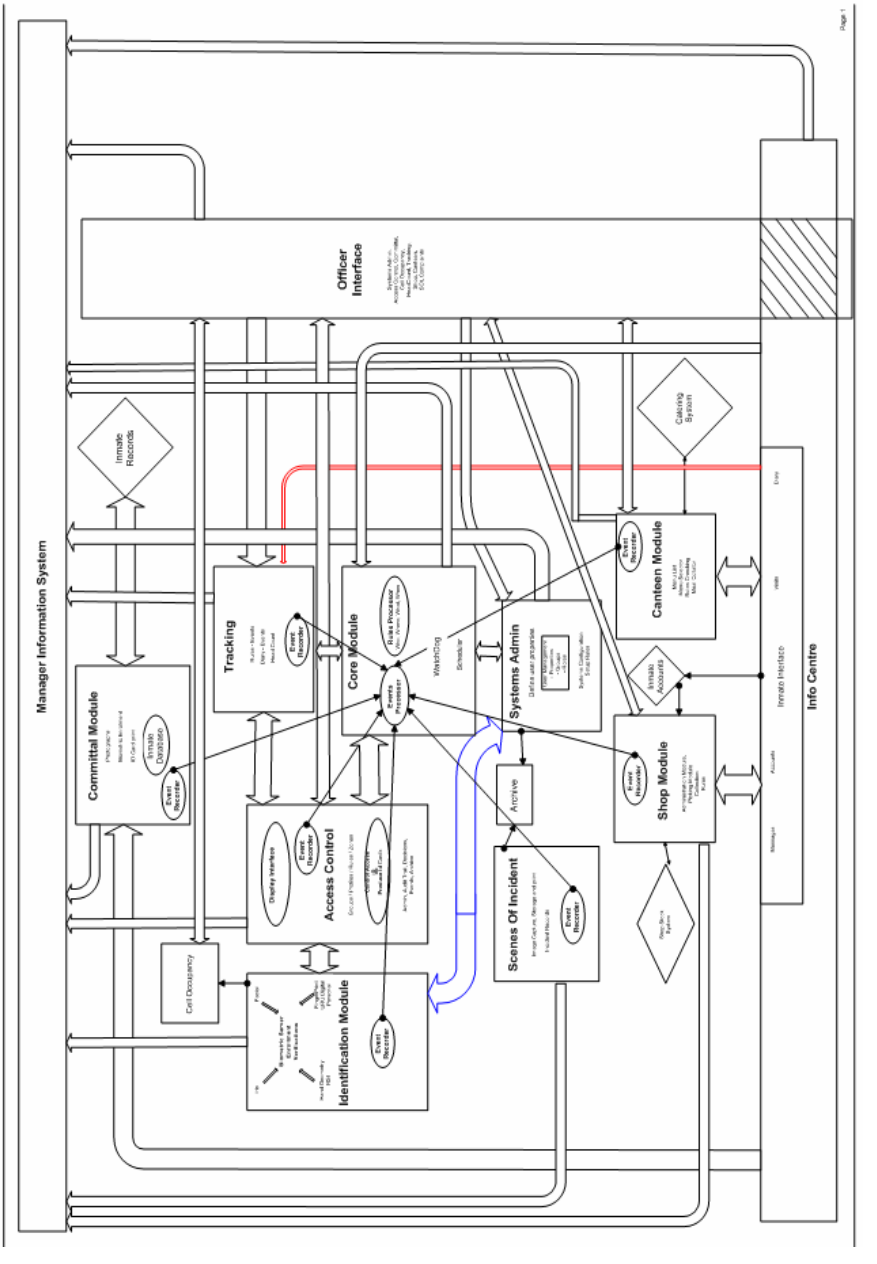


Fig. 1. Core Security System - High Level Architecture

Figure 1 illustrates the Survivable Secure System (SSS) module architecture. The following sub-sections briefly describe their roles.

3.1 Users of the System

There are three key user types in this system: inmates, officers and management. To achieve inmate co-operation and buy-in to the security system they are also provide with benefits. Through an inmate touch screen kiosk using biometrics to ensure confidential access to private information such as inmate record details.

An Officer's access to the system is through more conventional PC access to organize and administrate prisoner services for instance meal distribution, tuck shop orders and prisoner movements.

Management access to the system can improve efficiency, enhance security, and provide tools for trend analysis. One such benefit is the ability to provide direct information services to the prisoners and free up officers from the role of information providers.

3.2 Identification Module

The identification module represents the technology being used to identify people. The architecture supports the use of a range of biometric technologies, i.e, fingerprint, hand geometry, iris and facial recognition. A biometric Server is responsible for the management of the connected biometric readers and will have its own internal template database. The Hand readers and Fingerprint sensors communicate with the biometric server through the Hand reader Server and Fingerprint server. Through the servers, the Biometric Server will obtain a list of all attached devices.

3.3 Access Control Module

Linked to the Identification module is the Access Control Module. The Access Control Module will enable remote management of all doors and gates. Servers that interact with the Access Control module are briefly described below.

3.3.1 Biometric Server

The access control server (ACS) registers with the biometric server as the default application for all readers defined in the database. When a verification message is received it will decide if access should be granted to the individual requesting access. When the decision is positive an appropriate unlock message is sent to the door server. When the request is denied a message may be sent to the door server to light a red lamp to indicate to the user that they have been denied access. Decisions should be made using the current zone information, the allowed zones and any time element.

3.3.2 Enrolment Console

The ACS sends an alarm message to the console when a visitor, who has a vehicle attached to their pass, tries to enter the outside zone.

3.3.3 Control Console

The ACS sends zone update messages to the control console giving the number of people/vehicles in each zone. These messages are sent every time a person or vehicle enters or exits a zone. When a control console starts up it registers with the ACS in the same way as it does with the door server. The control console also sends messages to the ACS whenever an operator manually relocates a person or vehicle. These messages are treated in the same way as in they will cause zone update messages to be sent out. The relocation messages can also relate to normal vehicle movement using the card swipes and the operator responding to the vehicle move.

3.3.4 Card Reader Server

When the ACS starts up it registers with the card reader servers that are defined in the database. When a card is swiped at a reader the card number is sent to the ACS. The ACS looks the card number up in the database and if the card is a vehicle card it will send a message to the appropriate PDA Console Server or control console.

3.3.5 PDA Console Server

The PDA Console displays information relating to vehicles wanting to gain access to zones.

3.4 Committal Module

The committal console will allow staff to add basic inmate details to the system along with the inmates photograph. Once the inmate has been added the operator can print a set of identification cards for the inmate. After committal, the operator can use the administration console to add additional more information can be added at the administration console where more cards can be printed if required.

3.5 Tracking

Rules will be related to tracking – (Diary) i.e. to keep track of Inmates, Staff, Visitors etc. Access Control will also relate to Tracking, as it defines where they are (zones) and what they can do (properties).

3.6 Core Module

The Core Module will contain the main ‘Events Processor’. This will hold the events produced from each module. Also contained within the Core Module is the ‘Rules Processor’ which will define Who – (Visitors, Inmates) should be Where – (Cell, Visiting Room) and at what time.

3.7 Cell Management

This will allow for the definition of house blocks, wings and cells. It allows the allocation of inmates to cells and keeps records of each movement. A graphical map is used to display house blocks, wings and cells as well as occupation.

3.8 Info Centre

Info Centre is a touch screen interface to provide general information (for all inmates) and personalised information (tailored for each inmate).

- Personal Information e.g. Release Date, bullying etc.
- General Information e.g. Rules and Regulations, Training etc.
- Requests for products and services

3.9 Officer Module

Each House Officer will have an interface to the system, which will allow them to monitor the activities of Inmates they are in charge of. This will reduce the time involved in the administrative duties an Office must carry out for his wing, e.g. Tuck shop orders, Canteen ordering, education etc.

3.10 MIS Module

The Management Information System will provide information to Management in a Prison that will help them plan for the future. This can be achieved by identifying trends in the data e.g. fights, queues at specific location.

It can also assist in making appropriate decisions about individual offenders. Access to the inmates' criminal history will be available, length of stay expected. This module could also be used to track inmate populations.

3.11 Scenes of Incidence

Scenes of Incidence (SOI) is used to capture images. SOI is related to Secure Image Processing System (SIPS).

SIPS is used to store images on a secure central database. There are two main components, Prisoner identification and scenes of incidence.

3.12 Watchdog

Watchdog is a service that monitors a specific list of devices/programs and communicates this status to registered consoles. The service will also accept commands from consoles allowing the monitored devices to be restarted.

The service maintains a list of monitored devices in the system SQL database and allows consoles to restart selected devices.

This program is designed to work with any SNMP compatible device. The aim of the program is to monitor the status of specific ports on SNMP Netgear switches on the network. It will also receive traps from the switch.

3.13 Systems Administration

This covers the administration of several categories of people, vehicles, zones, Card Templates and reports on people.

4 Core Survivable Security System Planes Examples

Table 3 depicts some of the Autonomicity / Selfware / Autonomy (self-directing and self-managing) plane activities within the Core Survivable Security System.

Table 3. Self-Directing & Self-Managing Systems Architecture Planes for Core Systems

Plane	***Core Survivable Security System***
Autonomous (over a time period)	Pattern and trend analysis of inmate behaviour
Selfware (day-2-day)	Bio recognition and gate access
Autonomic (reflex reaction)	Lock down and server reboot from watchdog.

It is key that a Self-* System operates with several speed loops of control – reflex operations, normal activity and a slower loop detecting patterns both reflective (monitoring its own self-* behaviour to ensure correct function) and user behaviour over time. These are all required to ensure a survivable system from necessary instant reflex self-healing operations if a server hangs to long term identification of intermittent and incorrect self-* behaviour.

5 Conclusion

A key point being made in this paper is the need for dynamics within responses and multiple loops of control; some fast (and possibly imprecise but immediate to overcome component faults and avoid system failure) and some slow (due to consideration over time of behaviour, both system and users).

These dynamics and multiple loops of control have been presented within three architectural planes: autonomic / selfware / autonomous. A 'reflex reaction' in the autonomic layer is based on the need for a system to respond more quickly than it can respond after detailed analysis and planning of a comprehensive response – simply as that timeframe may be too late.

The paper detailed a deployed Survivable Security System also referred to as Biometric Identification and Tracking System (BITS) used in law enforcement and correction centers. The system operates across these three planes, taking autonomic, automatic and autonomous actions to ensure failure does not occur at the system level. As such, the survivable property of systems can be assisted by utilizing autonomic mechanisms [21].

Acknowledgements

The development of this paper was supported by a Knowledge Transfer Partnership project funded by the UK Government's Department of Trade and Industry (DTI). Core Systems' development project is partly supported by InvestNI. The wider

context of the Autonomic Systems research is supported at University of Ulster by the Computer Science Research Institute (CSRI) and the Centre for Software Process Technologies (CSPT), funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

References

1. P. Horn, “Autonomic computing: IBM perspective on the state of information technology,” IBM T.J. Watson Labs, NY, 15th October 2001.
2. R. Sterritt, “Towards Autonomic Computing: Effective Event Management”, Proc. IEEE/NASA SEW, Greenbelt, MD, Dec. 2002.
3. JO Kephart, DM Chess. “The Vision of Autonomic Computing”, Computer, 36(1):41–52, 2003.
4. R. Sterritt, "Autonomic Computing", Innovations in Systems and Software Engineering, Vol. 1, No. 1, Springer, pp 79-88, 2005
5. R. Sterritt, D.F. Bantz, “PAC-MEN: Personal Autonomic Computing Monitoring Environments,” Proc IEEE DEXA 2004 Workshops - 2nd Int. Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 04), Zaragoza, Spain, Aug. 30 – 3 Sept., 2003.
6. G. Kaiser, J. Parekh, P. Gross, G. Valetto, “Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems,” Autonomic Computing Workshop – IEEE 5th Int. Active Middleware Workshop, Seattle, USA, June 2003.
7. E. Lupu, et al., EPSRC AMUSE: Autonomic Management of Ubiquitous Systems for e-Health, 2003.
8. Hu Jun, Gao Ji, Huang Zhongchao, Liao Beishui, Li Changyun, Chen Jiujun, A New Rational Model for Agent for Autonomic Computing, 2004 IEEE International Conference on Systems, Man and Cybernetics, pp 5531-5536, 2004.
9. V. Tamma, I. Blacoe, B. Lithgow Smith, M. Wooldridge “Introducing autonomic behaviour in semantic web agents”, Proceedings of the Fourth International Semantic Web Conference (ISWC), Galway, Ireland, November 2005.
10. American Heritage Dictionary of the English Language, 4th edition, 2005.
11. R. Sterritt, M.G. Hinchey, (Apr 2005) "Birds of a Feather Session: “Autonomic Computing: Panacea or Poppycock?”", Proceedings of IEEE Workshop on the Engineering of Autonomic Systems (EASe 2005) at 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005), Greenbelt, MD, USA, 3-8 April, 2005, Pages 335-341
12. H.A. Simon, (1967), ‘Motivational and emotional controls of cognition’. Reprinted in Models of Thought, Yale University Press, 29–38, 1979.
13. D.A. Norman, A. Ortony, D.M. Russell, “Affect and machine design: Lessons for the development of autonomous machines”, IBM Systems Journal, Vol. 42, No.1, pp 38- 44, 2003.
14. A. Sloman, M. Croucher, “Why Robots Will Have Emotions”, Proc. 7th International Joint Conference on AI, pp 197-202, Vancouver, 1981.
15. A. Sloman, “Review of: Rosalind Picard’s Affective Computing, MIT Press, 1997.” AI Magazine 1999.
16. D Clark, C Partridge, JC Ramming, JT Wroclawski, “A Knowledge Plane for the Internet”, Proc. Applications, technologies, architectures, and protocols for computer communication, Karlsruhe, ACM SIGCOMM 2003

17. SM Lewandowski, DJ Van Hook, GC O'Leary, JW Haines, LM Rossey, “SARA: Survivable Autonomic Response Architecture”, DARPA Information Survivability Conference and Exposition II Proceedings, Vol. 1, pp. 77-88, June 2001.
18. W. Truszkowski, L. Hallock., C. Rouff, J. Karlin, J. Rash, M.G. Hinchey, R. Sterritt, “Autonomous and Autonomic Systems with Applications to NASA Intelligent Spacecraft Operations and Exploration Systems”, Springer, *in preparation, due 2006.*
19. IBM, “An architectural blueprint for autonomic computing”, 2003.
20. R Sterritt, DW Bustard, "Autonomic Computing: a Means of Achieving Dependability?", Proc IEEE Int. Conf. on the Engineering of Computer Based Systems (ECBS'03), Huntsville, Alabama, USA, April 7-11 2003, pp 247-251.
21. R. Sterritt, G. Garrity, E. Hanna, P. O'Hagan, “Autonomic Agents for Survivable Security Systems", 1st IFIP Workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005) at EUC'05, Nagasaki, Japan, 6-9th December, in "LNCS 3823", *Forthcoming* Dec. 2005

Author Index