Lecture Notes in Artificial Intelligence 2577

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Paolo Petta Robert Tolksdorf
Franco Zambonelli (Eds.)

Engineering Societies in the Agents World III

Third International Workshop, ESAW 2002
Madrid, Spain, September 16-17, 2002
Revised Papers

Springer

Series Editors

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

Volume Editors

Paolo Petta
Austrian Research Institute for Artificial Intelligence
Intelligent Agents and New Media Group
Schottengasse 3, 1010 Vienna, Austria
E-mail: paolo@oefai.at

Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik, AG Netzbasierte Informationssysteme
Takustr. 9, 14195 Berlin, Germany
E-mail: tolk@inf.fu-berlin.de

Franco Zambonelli
Università di Modena e Reggio Emilia
Dipartimento di Scienze e Metodi dell'Ingegneria
Via Allegri 13, 42100 Reggio Emilia, Italy
E-mail: franco.zambonelli@unimo.it

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): I.2.11, I.2, C.2.4, D.1.3, D.2.2, D.2.7, D.2.11

ISSN 0302-9743
ISBN 3-540-14009-3 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN: 10872344 06/3142 5 4 3 2 1 0

Preface

The characteristics of software systems are undergoing dramatic changes. We are moving rapidly into the age of ubiquitous information services. Persistent computing systems are being embedded in everyday objects. They interact in an autonomous way with each other to provide us with increasingly complex services and functionalities that we can access at any time from anywhere. As a consequence, not only do the numbers of components of software systems increase; there is also a strong qualitative impact. Software systems are increasingly made up of autonomous, proactive, networked components. These interact with each other in patterns and via mechanisms that can hardly be modeled in terms of classical models of interaction or service-oriented coordination. To some extent, future software systems will exhibit characteristics making them more resemblant of natural systems and societies than of mechanical systems and software architectures.

This situation poses exciting challenges to computer scientists and software engineers. Already, software agents and multi-agent systems are recognized as both useful abstractions and effective technologies for the modeling and building of complex distributed applications. However, little is done with regard to effective and methodic development of complex software systems in terms of multi-agent societies. An urgent need exists for novel approaches to software modeling and software engineering that enable the successful deployment of software systems made up of a massive number of autonomous components, and that allow us to control and predict their behaviour. It is very likely that such innovations will exploit lessons from a variety of different scientific disciplines, such as sociology, economics, organization science, modern thermodynamics, and biology.

The sequel to successful editions in 2000 in Berlin and 2001 in Prague, ESAW 2002 remained committed to the use of the notion of multi-agent systems as the seed for animated, constructive, and highly interdisciplinary discussions about technologies, methodologies, and tools for the engineering of complex distributed applications. While the workshop placed an emphasis on practical engineering issues, it also welcomed theoretical, philosophical, and empirical contributions, provided that they clearly documented their connection to the core applied issues.

The organizers received 35 papers that underwent a strict scientific peer review for quality, with three reviews per paper. This process led to the selection of 20 papers for presentation at the workshop. The accepted papers were reviewed again for this volume to ensure the integration of reviewers' comments and issues that arose from discussions at the workshop.

The discussions at the workshop were stimulated by an invited presentation of Marco Dorigo, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle Université, IRIDIA, Université Libre de Bruxelles on “Ant Algorithms, Self-organization and Stigmergy”.

The organizers want to thank Europe's IST-funded Network of Excellence for agent-based computing, AgentLink II, the Austrian Society for Artificial Intelligence (ÖGAI) and the Universidad Rey Juan Carlos, Madrid for financial support in realizing the workshop.

Madrid, September 2002

Paolo Petta
Robert Tolksdorf
Franco Zambonelli

Organization

Workshop Organizers

Paolo Petta	Austrian Research Institute for Artificial Intelligence, Vienna, Austria
Robert Tolksdorf	Institut für Informatik, Freie Universität Berlin, Germany
Franco Zambonelli	Dip. di Elettronica, Informatica e Sistemistica, Univ. Bologna, Reggio Emilia/Modena, Italy

Local Organizing Chair

Sascha Ossowski	Grupo de Investigación en Inteligencia Artificial, Univ. Rey Juan Carlos, Madrid, Spain

Programme Committee

Federico Bergenti	Dip. di Ingegneria dell'Informazione, Univ. degli Studi di Parma, Italy
Jeffrey Bradshaw	Inst. for Human and Machine Cognition, Univ. of West Florida, USA
Cristiano Castelfranchi	Inst. of Psychology, CNR, Italy
Paolo Ciancarini	Dip. di Scienze dell'Informazione, Univ. di Bologna, Italy
Helder Coelho	Dept. of Informatics of the Faculty of Sciences, Univ. of Lisbon, Portugal
Keith Decker	Dept. of Computer and Information Sciences, Univ. of Delaware, Newark, DE, USA
Paul Davidsson	Dept. of Software Engineering and Computer Science, Blekinge Inst. of Technology, Karlskrona, Sweden
Bruce Edmonds	Centre for Policy Modelling, Manchester Metropolitan Univ., Manchester, UK
Rino Falcone	Inst. of Psychology, CNR, Italy
Tim Finin	Computer Science and Electrical Engineering Dept., Univ. of Maryland, Baltimore County, MD, USA
Stephan Flake	Cooperative Computing & Communication Lab, Innovation Center of Siemens and Univ. Paderborn, Germany
Martin Fredriksson	Dept. of Software Engineering and Computer Science, Blekinge Inst. of Technology, Karlskrona, Sweden
Marie-Pierre Gleizes	Inst. de Recherche en Informatique de Toulouse, Univ. Paul Sabatier, Toulouse, France
Rune Gustavsson	Dept. of Software Engineering and Computer Science, Blekinge Inst. of Technology, Karlskrona, Sweden

Nicholas Jennings	Intelligence, Agents and Multimedia Group, Dept. of Electronics and Computer Science, Univ. of Southampton, UK
Paul Kearney	Intelligent Agents, BT exact, UK
Matthias Klusch	German Research Center for Artificial Intelligence – DFKI GmbH, Saarbrücken, Germany
Yannis Labrou	Powermarket, Inc., Belmont, CA, USA
Lyndon C. Lee	Intelligent Agents, BT exact, UK
Michael Luck	Intelligence, Agents and Multimedia Group, Dept. of Electronics and Computer Science, Univ. of Southampton, UK
Scott Moss	Centre for Policy Modelling, Manchester Metropolitan Univ., Manchester, UK
Pablo Noriega	Laboratorio Nacional de Informática Avanzada, A.C, Xalpa, Vera Cruz, México
Andrea Omicini	Dip. di Elettronica, Informatica e Sistemistica, Univ. di Bologna, Cesena, Italy
Sascha Ossowski	Grupo de Investigación en Inteligencia Artificial, Univ. Rey Juan Carlos, Madrid, Spain
H. Van Dyke Parunak	Altarum Inst., Ann Arbor, MI, USA
Michal Pechoucek	Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical Univ., Prague, Czech Republic
Jeremy Pitt	Intelligent and Interactive Systems, Electrical and Electronic Engineering Dept., Imperial College of Science, Technology and Medicine, London, UK
Agostino Poggi	Dip. di Ingegneria dell'Informazione, Univ. degli Studi di Parma, Italy
John R. Rose	Dept. of Computer Science and Engineering, College of Engineering, Univ. of South Carolina, Columbia, USA
Onn Shehory	IBM Haifa Research Laboratories, Haifa, Israel
Christophe Sibertin-Blanc	Inst. de Recherche en Informatique de Toulouse, Univ. Paul Sabatier, Toulouse. France
José M. Vidal	Dept. of Computer Science and Engineering, College of Engineering, Univ. of South Carolina, Columbia, USA
Gerhard Weiß	Theoretische Informatik und Grundlagen der Künstlichen Intelligenz, Inst. für Informatik, Technische Univ. München,Germany
Bin Yu	Information Technology and Engineering, College of Engineering, Carolina State Univ., Raleigh, USA

Table of Contents

Views

Models

Engineering

Modelling and Design

A Discussion of Two Major Benefits of Using Agents in Software Development

Federico Bergenti [1,2]

[1] AOT Lab
Parco Area delle Scienze 181/A, 43100 Parma, Italy
bergenti@ce.unipr.it
http://aot.ce.unipr.it
[2] FRAMeTech
Via San Leonardo 1, 43100 Parma, Italy
bergenti@frame-tech.it
http://www.frame-tech.it

Abstract. Agent-oriented software engineering has not yet solved the basic problem of *why* we should use agents to build our software system. Why is it convenient to use agents instead of more mature technologies like, for example, software components? This paper addresses this issue and compares a BDI-like agent model with well-known component models like Enterprise JavaBeans, CORBABeans and .NET components. The two major results of such a comparison are: *(i)* agents are more reusable and more composable than components, and *(ii)* agents allow to describe systems at a higher level of abstractions than components. This work is not meant to be conclusive; rather it intends to start a debate on these and related topics.

1 Introduction

The ultimate objective of agent-oriented software engineering (AOSE) is to provide all necessary tools to engineer agent-based systems. Such tools include, but are not restricted to, methodologies and development frameworks. Nowadays, AOSE has partially accomplished its task: we have some good methodologies and some good development frameworks also. Yet, all this work may not be sufficiently rewarded if the AOSE community would continue avoiding the very basic question of *why* we should use agents to build software systems. Why shall we employ agent-based technologies instead of choosing any other technology? Areńt available technologies sufficient? These, and many other similar questions that may come to your mind, are a pain for us interested in agents. Put simply, we have no *real* answer to them. The debate on the differences between agents and objects exacerbated the situation: the supposed advantages over the object-oriented approach seem poor and the results of some comparisons are simply wrong. Just to cite a common mistake: people often forget that the metaobject protocol [9] was introduced in mid-80ís and they keep saying that one

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 1-12, 2003.

big difference between agents and objects is that agents send messages while objects invoke methods.

This paper is meant to be a first step in the direction of answering the very basic question of why agents are a good paradigm for software development. I proceed toward this end in the following way: first, I compare agents with state-of-the-art technology for software development, i.e., software components like Enterprise JavaBeans (EJBs) [15], CORBABeans [14] and .NET components [6]. Then, I elaborate on the first main result of such a comparison: agents are more reusable and more composable than components. Finally, I discuss the second main result of the comparison: agents raise the level of abstraction with respect of component-based development. I formalize this new level of abstraction as a new system level and I briefly summarize its properties.

Before approaching the main topic of the paper, I need to briefly discuss the agent model that I use. I have to address this issue because components are a concrete object model and I need a concrete agent model to draw a reasonable comparison. I chose the agent model the ParADE [2] introduces. Such a model is a BDI-like incarnation and it was designed to focus on the main characteristics that agents share with components, i.e., reusability, composability and interoperability. It is common to emphasize this design focus using the word *agentware* for software build through the composition of agents of this sort. For the sake of brevity, I do not describe the model in much detail and I restrict the details to what I need to support the comparison with components. It is worth noting that the results of this work also apply to other BDI-like agent models.

2 A Model of Agentware

Agentware is software made through the composition of a set of interacting agents. In the ideal world, agents are taken from a repository of commercial off-the-shelf (COTS) agents and the multiagent system is composed and reconfigured on the fly to cope with varying conditions. Agents are known to the multiagent system by means of a unique identifier and their state is characterized through beliefs and intentions. You may want to use the word goals for such intentions because there is no difference between the two abstractions here. In the attempt to support different agent models, I do not go formal and I refrain from defining the properties of beliefs and intentions.

In addition to a unique identifier and a mental state, an agent has capabilities. These are described in terms of what the agent can do, i.e., the possible outcomes of its actions, and how the agent can interact with other agents.

Agents communicate through message passing and communication allows them to exchange representations of theirs and others beliefs, intentions and capabilities. Normally, beliefs, intentions and capabilities are represented through logic formulae and the ultimate purpose of communication is to exchange logic formulae that incorporate modalities for beliefs, intentions and capabilities. This may seem a rather extreme approach to agent-based communication, but it simply generalizes the available work on agent communication languages (ACLs). Letís take the FIPA ACL [7] as an example: it defines performatives together with feasibility preconditions and rational effects. When an agent receives a message, it can assert that the feasibility precondition holds for the sender and that the sender is trying to achieve the corresponding rational effect. This is basically a rather knotty way to let the receiver know what is going on in the senderís mental state. The advantage of using a structured ACL instead of a more natural exchange of representations of beliefs, intentions and capabilities simplifies the development of reactive agents capable of complex interactions.

The semantics that we normally use with most popular ACLs rely on heavy assumptions on the capabilities of agents. For example, the FIPA ACL requires an agent to reason on the receiverís mental state before sending any message. This seems too strong a requirement if we want to use agents in large applications where many, and possibly unforeseen, interactions occur. This is the reason why ParADE introduces a FIPA-like ACL with minimalist semantics. Such a language provides an operational means for agents to exchange representations of beliefs, intentions and capabilities. Table 1 shows some performatives of the ParADE ACL together with their semantics. The semantics is modeled as the effect that the sender wishes to achieve when sending the message.

The ParADE ACL provides a means for exchanging complex logic formulae through simple messages as the receiver can assert what the sender is intending. This is sufficient to ensure semantic interoperability [2] without revealing too much details of the mental state of the sender, i.e., without breaking encapsulation.

Strictly speaking, the particular sort of ACL is not part of the agent model that I am presenting. Anyway, I outlined the ParADE ACL to ground some considerations that I am going to make in the following section.

Isolated messages are not sufficient to allow agents to communicate fruitfully. The classic example is the case of an agent requesting another agent to perform an action. The first problem is that messages are asynchronous: there is no guarantee that the receiver would act in response to a message, i.e., there is no guarantee that the receiver would actually perform the action. The second problem is that the semantics of a single message might not be sufficient to express application-specific constraints. The semantics of *request* does not impose the receiver to communicate to the sender that the requested action has been actually performed. The sender might hang indefinitely while waiting for the receiver to tell it that the action has been performed.

In order to support fruitful communication, the agent model that I am sketching here provides *interaction laws*. These are rules that an agent *decides* to adopt to govern its interactions with other agents. The interaction laws that an agent decides to follow are part of its capabilities and they are published. The following interaction law is sufficient to solve the classic problems of *request(s, r, a)*:

Table 1. Semantics of some performatives of the ParADE ACL

Message	Semantics
Inform(s, r, p)	$I_s B_r p$
Achieve(s, r, p)	$I_s I_r p$
QueryRef(s, r, p(x))	$I_s B_s[B_r q \wedge \text{unifies}(p(x), q)]$
Request(s, r, a)	$\|\text{achieve}(s, r, \text{done}(a))\|$
Agree(s, r, a)	$\|\text{inform}(s, r, I_s\text{done}(a))\|$
Refuse(s, r, a)	$\|\text{inform}(s, r, \neg I_s\text{done}(a))\|$

$$
\begin{aligned}
B_s I_r done(a) &\leftarrow I_r done(a) \\
B_s \neg I_r done(a) &\leftarrow \neg I_r done(a) \\
B_s done(a) &\leftarrow B_r done(a)
\end{aligned}
\tag{1}
$$

The first rule states that if agent *r* intends to perform *a*, then *s* must know it. The second rule covers the opposite case: if *r* decides not to perform *a*, then *s* must know it. The third rule says that as soon as *r* comes to know that *a* has been done, then also *s* must know it.

An interaction law has a precondition and a termination condition. In the example above, the precondition for *r* is $I_s done(a)$, i.e., *r* received the request. The termination condition is $B_r done(a) \vee \neg I_r done(a)$. An agent starts applying an interaction law when the precondition is verified and stops as soon as the termination condition holds.

With the introduction of precondition and termination condition, interaction laws become an elegant and flexible way to describe interaction protocols. The example above is the interaction law that describes the FIPA request protocol[7]. Similar approaches to overcome well-known problems of the classic descriptions of interaction protocols are also available [13], but they do not exploit the semantics of the underlying ACL.

The interaction laws than an agent may adopt are linked to the possible roles it can play in the multiagent systems, and they may vary over time.

3 Comparison with Software Components

The comparison between agents and components starts from table 2. It shows some important component-oriented abstractions and associates them with agent-oriented counterparts.

Table 2 compares some of the abstractions that form the componentsí metamodel with the corresponding abstractions of the agentsí metamodel. This suggests that a complete comparison should take into account also the abstractions that are missing in table 2. Following this approach, I should take into consideration, e.g., the .NET metamodel [6] and compare it with, e.g., the SMART framework [10]. This approach seems to overkill the problem because many abstractions in these metamodels are too different and they are hardly comparable.

Table 2. Comparison between component-oriented and agent-oriented abstractions

Abstraction	Component-oriented	Agent-oriented
Communication	Task delegation	Task and goal delegation
Message	Requests for actions	ACL messages
Interaction with the environment	Events	Updates of beliefs
State	Properties and relations	Mental attitudes
Interactions between parties	Interfaces, Interface repository	Capabilities, Semantic matchmaker
Runtime	Application server	FIPA platform

3.1 Comparing Abstractions

In the following, I take the component-oriented abstractions identified in table 2 and compare them with their agent-oriented counterparts. I did not choose any ranking criteria yet and therefore I cannot say which of the two approaches is better.

Communication Model. The main difference between agents and components is in the mechanism they use to communicate. Agents use ACLs, like the one that I introduced in the previous section, while components use a metaobject protocol [9]. In the agent-oriented approach, a message is sent to transfer part of the senderís mental state to the receiver. A special case of this mechanism is when the sender endeavors to delegate a goal to the receiver, e.g., through the *achieve* performative. This special case, known as goal delegation [5], is the basic mechanism that agents use to delegate responsibilities.

The componentsí metamodel does not comprise an abstraction of goal and therefore components cannot exploit goal delegation; rather they use task delegation. Components achieve their (implicit) goals asking to other components to perform actions; agents achieve their (explicit) goals delegating them to other agents. This is the reason why I refer to the agent-oriented communication model as *declarative message passing*: agents tell other agents what to do without stating how to do it. On the contrary, I use *imperative message passing* for the component-oriented approach because components cannot say to another component what to do without also saying how to do it.

The possibility of using task delegation only is a strong limitation for components because goal delegation is a more general mechanism. First, task delegation is a special case of goal delegation: the delegated goal has the form *done(a)*, just like in the semantics of the *request* performative. Then, task delegation may inhibit optimizations. Consider, e.g., a component s with a goal g that needs component r to perform a_1 and a_2 to achieve it; s would ask to r to perform a_1 and then it would ask to perform a_2. As the two requests are not coupled though the underlying idea that s is trying to achieve g, r cannot exploit any possible cross-optimization between a_1 and a_2.

If s and r were two agents instead of two components, s would simply delegate goal g to r and then r would decide autonomously the way to go, e.g., it would decide to perform a_1 and a_2. This approach couples a_1 and a_2 through g thus enabling r to perform cross-optimizations between a_1 and a_2.

Interaction with the Environment. The environment is a structural part of the agentsí metamodel. Agents execute in an environment that they can use to acquire knowledge. Agents can measure the environment and they can receive events from it. In both cases, agents react to any change in the environment through changes in their mental state. This is radically different from the component-oriented approach where the environment communicates with components through reified events. Components react to reified events after constructing a relation with them.

The component-oriented approach *seems* to better respect encapsulation than the agent-oriented approach: the state of the component is changed only when the component itself decides to change it in reaction to an event. If we go more in detail, we see that also the agent-oriented approach respects encapsulation. Agents have reasoning capabilities that are the actual responsible of any change in the mental state. Any direct push of knowledge from the sensors to the mental state is ruled through reasoning and the mental state remains encapsulated.

State Representation. Both agents and components are abstractions that comprise a state, but they have very different approaches to describe and to expose it. The state of a component is represented though a set of properties that other components can manipulate; such properties are the attributes of the component itself. In addition to properties, the state of a component includes its relations with other components. Such relations represent what the component knows about other components and how it relates to them.

In the model I introduced in the previous section, the state of an agent is represented though a set of beliefs, intentions and capabilities. The main differences with the component-oriented approach are:

1. Agents have an explicit representation of their goals;
2. Agents have explicit knowledge of their environment and not only of other agents;
3. Except for a unique identifier, agents do not have properties, they only have relations with other agents and with entities in the environment;
4. Agents may use deduction to come to know more than what other agents told them and more than what they measured.

It is worth discussing the last point a bit. Components can deduce the value of a property using application-specific mechanisms. The difference is that properties and relations are not easily structured in a logic framework and it is difficult to use general-purpose reasoning techniques with them; any reasoning is hardcoded in the component itself.

Interaction between Parties. The different communication mechanisms influence how agents and components open themselves to the outer world. Components use interfaces to enumerate the services they provide and to tell clients how to get in contact with them. Sophisticated component models equip interfaces with preconditions, post-conditions and invariants to support design by contract [11]. Anyway, the important issue of providing a description of the semantics of the services directly in the interface is still open.

The agent-oriented approach eliminates interfaces and provides agents with capabilities that describe what the agent can do, i.e., the possible outcomes of its actions, and how the agent can interact with other agents, i.e., the interaction rules it can adopt. The use of capabilities instead of interfaces has the advantage that we can easily describe the semantics of the services that an agent offers using:

1. High-level, mentalistic abstractions, i.e., beliefs and intentions;
2. The model of the environment, i.e., entities and their relations.

Runtime Environment. FIPA started its work in 1996 through the definition of a runtime environment to support interoperable multiagent systems. This work is valuable and today the agent-oriented counterpart of an application server is a FIPA platform, like JADE [1] and LEAP [3]. Actually, application servers are huge pieces of software that offer enterprise features that available FIPA platforms do not provide yet, e.g., transactional persistency and security. FIPA has accepted such a limitation of available implementations and it is going in the direction of creating an abstract architecture of its agent-oriented runtime. Such an architecture can be concretely instantiated using an available FIPA platform or any other runtime support, e.g., an application server. This choice saves both worlds and it seems one of the most reasonable. Anyway, it is worth noting that the very naÔve approach of encapsulating an agent into a component so to run it in an application server has some drawbacks. The most remarkable one is that the threading model that the application server imposes to components may not be compatible with agents. Basically, the developer must choose between loosing some enterprise feature, e.g., fault tolerance and transparent scalability, and implementing only reactive agents. Some middleware providing a reasonable compromise are already available, e.g., Caribbean [18] and EJB 2.0 allow asynchronous messaging in EJBs.

3.2 Agents Improve Reusability and Composability

The comparison that I sketched above enumerates differences and similarities between components and agent, but it does not state whether an approach is better than the other because I did not adopt any ranking criteria. In this section, I consider two of the most important features of a development technology, i.e., reusability and composability, and I evaluate how and when agents are better than components from these points of view. As noted above, agents:

1. Can use goal delegation instead of task delegation only;
2. Use ACLs that exploit the agent model to give a semantics to interactions;
3. Have a mental state and can use reasoning to generate knowledge from events and messages.

These properties improve the reusability and the composability of agents with respect of components.

Goal delegation partially decouples the agent that delegates a goal from the delegate agent because the two are no longer coupled through a sequence of actions, but only though a shared goal. In concrete terms, this solves the *interface mismatching* problem, i.e., the problem of substituting a component with another component that offers the same services through a different interface. Client components are perfectly fine with the services of this new component, but they cannot actually use them because they do not know how to access the services. Agents solve this problem because the client agent simply delegates the goal, and then the delegate agent autonomously decides what to do to achieve it. Unfortunately, the interface mismatching problem is not completely solved because agents may support different ontologies, but this is an easier problem especially if we implement open systems that deal with, so called, standard ontologies.

There are other similar problems concerning reusability and composability that goal delegation solves. I am not aware of any catalog of these problems and the enumeration and relative solution of all of them is out of the scope of this work.

Other improvements in terms of reusability and composability come from the agent-oriented approach to communication. It turns many common tasks, e.g., informing and querying, into application specific only in the sense that they depend on the ontology[1]. Components implement informing and querying through properties and relations and this couples communication with the information model, i.e., with the component-oriented counterpart of the ontology, and with the semantics of get/set messages. Agents remove the dependency on get/set messages and promote reusability and composability, provided that the ontology remains the same.

The logic-based model of mental states allows using deduction and means-end reasoning to infer knowledge. This means that many messages and events can be turned to few common situations. The concrete behavior of the agent is encoded only for a limited set of situations while allowing it to react to a wide range of events and messages. This promotes reusability and composability because the behavior of the agent is partially decoupled from the concrete messages and events that it can handle.

The three characteristics of agents that I have just discussed account for their superiority in terms of reusability and composability. If we try to generalize this result, we can see that their superiority is perfectly reasonable because agents support *semantic composability*. Agents rely on a common executor model, i.e., the agent model, that moves most of the semantics of the composition to the semantics of the ACL. This is exactly the opposite of what components do; the component model shifts most of the semantics of the composition to the semantics of the action that a component executes in reaction to a message or an event. The latter is syntactic composability because the semantics of communication does not exploit a common model of the executor.

If we adopt a shared and accepted ACL, e.g., the FIPA ACL, we can say that agents are semantically interoperable while components are only syntactically interoperable.

4 Introducing the Agent Level

People began enumerating the benefits of working at a high level of abstraction in the early days of computer science. When computers could only be programmed in assembly language people felt the urge of higher-level abstractions, e.g., types and procedures. Today, the component-oriented approach provides high-level abstractions, e.g., messages and events, and any further raise of the level of abstraction may seem a poor result. Actually, the level of abstraction that developers use impacts heavily on any figure we may evaluate to measure the quality of a product, and it should not come as a surprise that any increase of the level of abstraction drastically increases the quality of the final product.

In order to show that the agent-oriented approach is at a higher level of abstraction than the component-oriented approach, I take Boochís words into account: *ì at any given level of abstraction, we find meaningful collections of entities that collaborate to achieve some higher level view [4].î* The meaningful collections of agent-oriented entities comprise beliefs, goals and capabilities, and they are obviously closer to human intuition than component-oriented counterparts, e.g., property, event and message. Following the standard approach, I formalize this idea defining a new system level.

[1] Messages depend also on the chosen content language, but this is not a real issue.

A system level [12] is a set of concepts that provides a means for modeling implementtable systems. System levels are layered in a stack and each level is mapped on one lower level. System levels abstract away for implementation details and higher levels provide concepts that are closer to human intuition and far away from implementation. This is why we say that system levels are levels of abstraction[2]. System levels are structured in terms of the following elements:

1. Components: atomic building blocks for building systems at that level;
2. Laws of compositions: laws that rule how components can be assembled into a system;
3. A medium: a set of atomic concepts that the system level processes;
4. Laws of behavior: laws that determine how the behavior of the system depends on the behavior of each component and on the structure of the system.

Newellís knowledge level [12] provides a better perspective for characterizing agents than that of any operational characterization. Besides, Newellís work dates back to 1982 and it did not take into account many of the features that we would like agents to have today. In particular, Newell was very much concentrated on the realization of single agents capable of exhibiting intelligent behavior. Nowadays, we are no longer interested in realizing a system in terms of a single monolithic agent, and we prefer to use agents as building blocks of multiagent systems. The intelligence results from the interaction of agents and it is an emerging property of the whole system. In Wegnerís words: *ì Interaction enhances* dumb algorithms *so they become* smart agents.*î*

Jenningsí proposal of the social level [8] goes in this direction trying to raise the level of abstraction of the knowledge level. This approach has the advantage of requiring minimal capabilities to agents, thus accommodating any sort of agent available in the literature. Nevertheless, it may remain too generic and its utility in building complex systems seems poor.

Taking into account the limitations of both Newellís and Jenningsí approaches, I propose a novel system level called, and I will justify later why, *agent level*. The agent level is thought as a reasonable compromise between Newellís position, that avoids dealing structurally with interactions, and Jenningsí view of interaction as the only relevant component of a system. At the agent level, single agents exhibit a rational behavior and they are described in terms of mental attitudes; they are also part of a society and they interact through an ACL. The agent level exploits the possibilities of rationality and it provides a way for distributing complexity across agents rather than centralizing intelligence into a single monolithic agent.

In order to define the agent level, I first enumerate the components that it comprises:

1. Agents, that interact to implement the functionality of the multiagent system;
2. Beliefs, that an agent has;
3. Intentions, that an agent is trying to achieve;
4. Actions, that an agent can perform on the environment where it executes;

[2] Newell explicitly stated the contrary, but it seems that he used the phrase ìlevel of abstractionî in a different way from how we use it in software engineering today.

5. Roles, that agents can play in the system;
6. Interaction laws, that rule how agents playing different roles can interact.

Having said that agents and mental attitudes are both components of the agent level should justify its name. This does not apply to the knowledge level, where the agent is the system you implement. It is also inappropriate for the social level, where agents are components with no particular characteristics, and the focus is on the society.

The laws of composition state that a multiagent system is composed of a set of interacting agents. Each agent is associated with a set of beliefs, a set of intentions, a set of actions and a set of roles. The interaction between agents is ruled through a set of interaction laws. Notably, this is not a limitation of autonomy because agents autonomously decide to play a particular role as a way to bring about their goals.

Exploiting the components of the agent level, we can define two familiar concepts that are no longer atomic:

1. Responsibility: an intention that an agent is constantly bringing about and that it does not drop even if it has already been achieved;
2. Capability: the post-conditions of an agentís action, i.e., what the agent is capable to achieve on its own, and the interaction laws it supports, i.e., how it can interact with other agents to achieve its intentions.

The medium of the agent level is the representation that each agent has about its and other agentsí beliefs, intentions and capabilities. An agent processes the medium of the agent level, i.e., comes to know other agentsí beliefs, intentions and capabilities, through interaction in order to achieve its goals. In my proposal, the principal means of interaction is communication and agents use an ACL to exchange representations of theirs and others beliefs, intentions and capabilities. Only agents produce and consume messages, and the computation of the multiagent system results from the messages that agents manipulate.

The law of behavior is an adaptation of Newellís principle of rationality: if an agent has knowledge that one of the actions in its capabilities will lead to satisfy one of its intentions or to respect one of its responsibilities, then the agent will select that action.

This principle is sufficient to connect the components of the agent level with the selection of an action without implying any mechanism through which this connection is implemented at lower system levels. It may be any technique of means-end reasoning or it may be hardcoded in the program of the agent. Table 2 summarizes the elements of the agent level.

Most of the discussions that Newell did about the knowledge level are still applicable to the agent level. In particular, just like the knowledge level, the agent level is radically incomplete and the agentís behavior cannot be designed without going to a lower level.

Table 3. Elements of the agent level

Element	Element of the Agent Level
System	Multiagent system
Components	Beliefs, intentions, actions, roles and interaction laws
Medium	Representations of beliefs, intentions and capabilities
Behavior	Principle of rationality

An agent-level model is not useful if we do not provide a means for concretely implementing the system. If we agree that the agent level can sit on top of the component level, then we need a mapping between elements of the agent level and systems at the component level. Generally speaking, any development tool that supports agent-level abstractions level actually implements this mapping. Just to cite one example, ParADE allows using UML diagrams to model a system at the agent level; then, it generates Java skeletons that are at the same level of abstraction of components.

5 Conclusions

The whole length of this paper is devoted to give reasonable argumentations on why developers should seriously take into account the possibility of using agents for their products. The section on the agent level is there to provide a scientific foundation to such argumentations. The two most important outcomes of these discussions are that agents:

1. Provide high level abstractions to developers;
2. Have some advantages over components in terms of reusability and composability.

The first point, i.e., working with high level abstractions, has well-known advantages but it also has a typical drawback: speed. In order to fully exploit the possibilities of agents, we need to implement an agent model similar to the one described in section 2. This model heavily relies on reasoning and agents of this kind are likely to be slow. Nowadays, this does not seem a blocking issue because speed is not always the topmost priority, e.g., time-to-market is often more stringent.

The second point, i.e., reusability and composability, is worth discussing a bit because components are advocated as the most composable and reusable technology. Actually, the improvement that agents obtain comes at a cost: speed, again. The use of goal delegation instead of task delegation requires, by definition, means-end reasoning and we face the reasonable possibility of implementing slow agents.

Fortunately, in both cases, the performances of agents degrade gracefully. We can choose how much reasoning, i.e., how much loss of speed, we want for each and every agent. In particular, we may use reasoning for agents that:

1. Are particularly complex and could benefit from high level abstractions;
2. We want to reuse and compose freely in possibly different projects.

On the contrary, we can fallback on reactive agents, or components, when we have an urge for speed. This decision criterion seems sound because the more an agent is complex and value-added, the more we want to reuse it and compose it with other agents. Moreover, reactive agents are perfectly equivalent to components and we do not loose anything using the agent-oriented approach instead of the component-oriented approach.

References

[1] Bellifemine, F., Poggi, A., Rimassa, G.: Developing Multi-agent Systems with a FIPA-Compliant Agent Framework. Software Practice and Experience 31 (2001) 103ñ128

[2] Bergenti, F., Poggi, A.: A Development Toolkit to Realize Autonomous and Inter-operable Agents. Procs. 5th Intíl Conference on Autonomous Agents, (2001) 632ñ639

[3] Bergenti, F., Poggi, A., Burg, B., Caire, G.: Deploying FIPA-Compliant Systems on HandHeld Devices. IEEE Internet Computing 5(4) (2001) 20ñ25

[4] Booch, G.: Object-Oriented Analysis and Design with Applications. Addison-Wesley (1994)

[5] Castelfranchi, C.: Modelling Social Action for AI Agents. Artificial Intelligence 103(1) (1998)

[6] European Computer Manufacturerís Association: Standard ECMA-335, Partition II Metadata Definition and Semantics. Available at `http://www.ecma.ch`

[7] Foundation for Intelligent Physical Agents. Specifications. Available at `http://www.fipa.org`

[8] Jennings, N. R.: On Agent-Based Software Engineering. Artificial Intelligence, 117 (2000) 277ñ296

[9] Kiczales, G., des RiviËres, J., Bobrow, D.G.: The Art of the Metaobject Protocol. MIT Press (1991)

[10] Luck, M., díInverno, M.: A Conceptual Framework for Agent Definition and Development. The Computer J. 44(1) (2001) 1ñ20

[11] Meyer, B.: Object-Oriented Software Construction. Prentice-Hall (1997)

[12] Newell, A.: The Knowledge Level. Artificial Intelligence, 18 (1982) 87ñ127

[13] Singh, M.P., Yolum, P.: Commitment Machines. Procs. 8th Intíl Workshop on Agent Theories, Architectures, and Languages (2001)

[14] Suhail, A.: CORBA Programming Unleashed. Sams (1998)

[15] Sun Microsystems: Enterprise JavaBeans Specification: Version 2.0. Available at `http://java.sun.com`

[16] Wegner, P.: Why Interaction is More Powerful than Algorithms. Communications of the ACM 40(5) (1997) 80ñ91

[17] Wooldridge, M. J., Jennings, N. R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Intíl. J. Autonomous Agents and Multi-Agent Systems 2 (1) (2000)

[18] Yamamoto, G., Tai, H.: Caribbean Application Developer Guide. Available at `http://alphaworks.ibm.com`

Signs of a Revolution in Computer Science and Software Engineering

Franco Zambonelli[1] and H. Van Dyke Parunak[2]

[1] Dip. di Scienze e Metodi dellíIngegneria ñ Univ. di Modena e Reggio Emilia
franco.zambonelli@unimo.it
[2] Altarum - 3520 Green Ct, Suite 300, Ann Arbor, MI 48105 USA
van.parunak@altarum.org

Abstract. Several characteristics distinguish today's complex software systems from "traditional" ones. Examples in different areas show that these characteristics, already the focus of agent-oriented software engineering research, influence many application domains. These characteristics will impact how software systems are modeled and engineered. We are on the edge of a revolutionary shift of paradigm, pioneered by the multi-agent systems community, and likely to change our very attitudes in software systems modeling and engineering.

1 Introduction

Computer science and software engineering are going to change dramatically. Scientists and engineers are spending a great deal of effort attempting to adapt and improve well-established models and methodologies for software development. However, the complexity introduced to software systems by several emerging computing scenarios goes beyond the capabilities of traditional computer science and software engineering abstractions, such as object-oriented and component-based methodologies.

The scenario initiating the next software crisis is rapidly emerging: computing systems will be everywhere, always connected, and always active [37]. Computer systems will be embedded in every object, e.g., in our physical environments, our clothes and furniture, and even our bodies. Wireless technologies will make network connectivity pervasive, and every computing device will be connected in some network, whether the "traditional" Internet or ad-hoc local networks. Finally, computing systems will be always active to perform some activity on our behalf, e.g., to improve comfort at home or to control and automate manufacturing processes.

This scenario does not simply affect the design and development of software systems *quantitatively*, in terms of number of components and effort required. Instead, there will be a *qualitative* change in the characteristics of software systems, as well as in the methodologies adopted to develop them. In particular, four main characteristics, in addition to the quantitative increase in interconnected computing systems, distinguish future software systems from traditional ones:

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 13-28, 2003.

- *situatedness*: software components execute in the context of an environment, can influence it, and can be influenced by it;
- *openness*: software systems are subject to decentralized management and can dynamically change their structure;
- *locality in control*: the components of software systems represents autonomous and proactive loci of control;
- *locality in interactions*: despite living in a fully connected world, software components interact accordingly to local (geographical or logical) patterns.

These characteristics commonly characterize multi-agent systems (MAS) [20]. Section 2 discusses them characteristics in detail and shows how, to different extents and with different terminology, various research communities (e.g., manufacturing control systems [7], mobile and pervasive computing [2, 40], sensor networks [14], groupware and enterprise infrastructures [22, 36], Internet [8] and P2P computing [32]) already recognize their importance and are adapting their models and technologies to them. Thus, our first contribution is to synthesize in a single conceptual framework several novel concepts and abstractions that are emerging in different areas without recognition of the basic commonalties, because of a lack of interaction and common terminology. This synthesis may suggest new applications for their broadly applicable MAS concepts.

Following this synthesis, Section 3 argues that integration of these concepts and abstractions in software modeling and design is not an incremental *evolution* of current models and methodologies, but a *revolution*, a radical change of paradigm [21] pioneered by the MAS community. This revolution will impact most research communities and will dramatically change how we conceive, model, and build "software components" and "software systems." Next generation software systems will no longer be modeled and designed as "mechanical" or "architectural" systems, but rather as "physical" or "intentional" systems. We try to identify the impact of such a change of paradigm in computer science and software engineering practices.

2 Whatís New?

The four characteristics identified in the introduction (situatedness, openness, local control, local interactions) affect most of todayís software systems.

2.1 Situatedness

Todayís computing systems are situated: they have an explicit notion of the environment in which components exist and execute, environmental characteristics affect their execution, and their components often explicitly interact with that environment.

Software systems have always been and will always be immersed in some sort of environment. For instance, the execution of a process in a multi-threaded machine is intrinsically affected by the dynamics of the multiprocessing system. Traditional modeling and engineering tries to mask the presence of the environment. In most the cases, specific objects and components ìwrapî the environment and model it in terms of a "normal" component, so that the environment in itself does not exist as a primary abstraction. Unfortunately, the environment in which components live and with which they interact does exist and may impact execution and modeling:

- Several entities with which software components may need to interact are too complex in their structure and behavior to enable a trivial wrapping.
- For a system whose goal is to monitor (sense) and control (affect) a physical or logical (computational) environment, masking the environment is not natural. Instead, providing an explicit consciousness may be a primary application goal.
- The environment can have its own dynamics, independent of a software systemís intrinsic dynamics. Wrapping the environment will introduce unpredictable non-determinism in the behavior of some parts of our applications.

For these reasons, both computer science and software engineering now tend to define a systemís execution environment as a primary abstraction, explicitly defining both the "boundaries" separating the software system from its environment and the reciprocal influences of the two systems. This approach both avoids odd wrapping activities needed to model each component of the external world as an internal application entity, and allows a software system to deal more naturally with its activities in the real-world environment it is devoted to monitor and control. In addition, explicit modeling of the environment and its activities makes it possible to identify and confine clearly the sources of dynamics and unpredictability (and, thus, of non-formalizability [39]), and concentrate on software components as deterministic entities that have to deal with a dynamic and possibly unpredictable environment.

Examples

Control systems for physical domains (e.g., manufacturing, traffic control, home care, health care) are often built by explicitly taking into account unpredictable environmental dynamics via specific event-handling (or similar) policies [15]. Similar problems arise in sensor and robot networks [19], where many components are spread in an unknown environment to explore and monitor it.

Mobile and pervasive computing recognizes the importance of context-awareness, e.g., applicationsí need to model environmental characteristics (e.g., those related to the sensed physical environment [16] or to the locations of specific components [30]) and environmental data (e.g., provided by an embedded infrastructure [3, 14]) explicitly, rather than implicitly in terms of internal object attributes.

Applications intended to be immersed in the intrinsically dynamic Internet environment are typically engineered by defining the boundaries of the system in terms of "application," including the new application components to be developed, and "middleware," the environmental substrate in which components will be embedded [8, 42]. Clearly identifying and defining such boundaries is a key point in web-application engineering. Several systems for workflow management and computer supported collaborative work are built around shared data space abstractions as the common execution environment for workflow processes and agents [36].

Finally, several promising proposals in distributed problem solving and optimization (i.e., works on ant-based colonies [27, 28]), exploit a dynamic virtual environment influencing the activities of distributed problem solver processes.

2.2 Openness

Living in an environment, perceiving it, and being affected by it intrinsically imply openness. The software system is no longer isolated, but becomes a permeable subsystem, whose boundaries permit reciprocal side-effects. This reciprocal influence between system and environment is often extreme and complex, making it difficult to identify boundaries clearly between the system and its environment.

In several cases, to achieve their objectives, software systems must interact with external software components, either to provide services and data, or to acquire them. More generally, different software systems, independently designed and modeled, are likely to "live" in the same environment and explicitly interact with each other. These open interactions call for common ontologies, communication protocols, and suitable broker infrastructures, to enable interoperability. However, this is only a small part of the problem.

- Simply enabling interoperability is not enough when software systems may come to life and die in an environment independently of each other, or when a software system (or its components) can explicitly move across different environments during its life. These characteristics introduce additional problems.
- When component in different software systems interact, and when components move across different environment, it may become hard if not impossible to identify clearly the system to which a component belongs. In other words, it becomes difficult to understand clearly the boundaries of software systems.
- When a component comes to life in an environment (or is somehow moved to a specific environment), it must somehow be made aware of what is around in the environment, what other components are there for interaction, and how it can interact with newly entering systems safely for both the systems and itself.
- Enabling components to enter and leave an environment in a fully free way and interact with each other may make it very hard to understand and control the overall behavior of these software systems and even of a single software system.

Due to these problems, computer science and software engineering are starting to consider the problem of not only modeling the environment in which systems execute, but also of understanding and modeling the dynamic changes in system structure. Also, the need to preserve coherency of the system despite openness may require identifying and enacting specific policies, intended to support the re-organization of the software system in response to changes in its structure, or to enact contextual laws on the activity of those components entering the system. The general aim is to increase the ability to control system execution despite its dynamic structural changes.

Examples

Control systems for critical environments, such as traffic control systems, public telephony services, health care systems, manufacturing systems, and sensor-based monitoring, run continuously, cannot be stopped, and sometimes cannot even be removed from the environment in which they are embedded. Nevertheless, these systems need to be updated, and their environment is likely to change frequently, with the addition of new physical components and, consequently, of new software components and software systems [37]. For all these systems, managing openness and a systemís ability to re-organize itself automatically (e.g., by establishing new effective interactions with new components and/or by changing the structure of the interaction graph [19, 32]) is of dramatic importance, as is the ability of a component to enter new execution contexts safety and effectively.

Mobility, whether of users, software, or devices, moves the concept of openness to the extreme, by making components actually move from one context to another during their execution, changing the structure of the software system executing in that context [8, 41]. This requires not only the ability of components to learn about their new context , but also the ability to organize and control component interactions in the context [29]. Such concepts are exacerbated in mobile ad-hoc networking, where interactions must be made fruitful and somehow controllable despite the lack of any intrinsic structure and the dynamics of connectivity [6]

Similar considerations apply to Internet-based and open distributed computing. There, software services must survive the dynamics and uncertainty of the Internet, must be able to serve any client component, and must also be able to enact security and resource control policy in their local context, e.g., a given administrative domain [8]. E-marketplaces are the most typical examples of this class of open Internet applications [25]. P2P systems stands to Internet applications as ad-hoc networking stands in mobile computing system: components must be allowed to fruitfully and properly interact without any pre-determined interaction structure and by surviving the dynamics of services and data availability [33].

2.3 Local Control

The "flow of control" concept has always been one of the key aspects of computer science and software engineering, at all levels, from the hardware level up to the high-level design of applications. However, when software systems and components live and interact in an open world, the concept of flow of control becomes meaningless.

Independent software systems have their own autonomous flows of control, and their mutual interactions do not imply any join of these flows. Therefore, the modeling and designing of open software not only makes the concept of "software system" rather vague, as discussed in Subsection 2.2, but it also makes the concept of "global execution control" disappear. This trend is exacerbated by the fact that not only do independent systems have their own flow of control, but also different components in a system may be autonomous in control. In fact, most components of today's software systems are active entities (e.g., active objects with internal threads of control, processes, daemons) rather than passive ones (e.g., "normal" objects, functions, etc.).

Having multiple threads of control in a system is not novel, and concurrent and parallel programming are well-established paradigms. However, the main motives for multiple threads in traditional concurrent and parallel programming are efficiency and performance. Most approaches try to limit the independence of these multiple threads as much as possible, via strict synchronization and coordination mechanisms, to preserve determinism and high-level control over applications. Today's autonomy of application components has different motives and has to be handled differently.

- In an open world, autonomy of execution enables a component to move across systems and environments without having to report back to (or wait for ack by) its original application.
- When components and systems are immersed in a highly dynamic environment whose evolution must be monitored and controlled, an autonomous component can be effectively delegated to take care of (a portion of) the environment independently of the global flow of control.
- Several software systems are not only made up of software components, but also integrate computer-based systems, which are by their very nature autonomous systems, and can be modeled accordingly.
- As the size of a software system increases, both performance and conceptual simplicity require delegating control to components. Coordinating a global flow of control among a very large number of components may become unfeasible. Autonomy then becomes an additional dimension of modularity [26].

Our concept of autonomy refers not only to those software components that are designed as autonomous, but also to those that can be perceived as autonomous.

Examples

Almost all modern software systems integrate autonomous components. Weak autonomy is the ability of a component to react to and handle events (e.g., graphical interfaces or embedded sensors). Strong autonomy implies that a component integrates an autonomous thread of execution, and can execute proactively. In most modern control systems, control is not simply reactive but proactive, realized via a set of cooperative autonomous processes or, often, via embedded complete computer-based systems interacting with each other [15] or distributed sensor nets [19].

The integration in complex distributed applications and systems of (software running on) mobile devices of any type can be tackled only by modeling them in terms of autonomous software components [29].

Internet based distributed applications are typically made up of autonomous processes, possibly executing on different nodes, and cooperating with each other, a choice driven by conceptual simplicity and by decentralized management rather than by the actual need of autonomous concurrent activities.

2.4 Local Interactions

Directly deriving from these three issues, the concept of "local interactions in a global world" is more and more pervading today's software systems.

By now, we have delineated a scenario in which software systems components are immersed in a specific environment, execute in the context of a specific (sub)system, and are delegated to perform some task in autonomy. Taken all together, these aspects naturally lead to a strict enforcement of locality in interactions. In fact:

- Autonomous components can interact with the environment in which they are immersed, by sensing and affecting it. If the environment is physical, the amount of the physical world a single component can sense and affect is locally bounded by physical laws. If the environment is logical, minimization of conceptual and management complexity still favor modeling it in local terms and limiting the effect of a single component on the environment.
- Components can normally interact with each other in the context of the software system to which they belong, that is, locally to their system. In open work, however, a component of a system can also interact with (components of) other systems. In these cases, minimization of conceptual complexity suggests modeling the component in terms of a component that has temporarily "moved" to another context, and that interacts locally in the new context [8].
- In an open world, components need some form of context-awareness to execute and interact effectively. For a component to be made aware of its context effectively (and efficiently), this context must necessarily be locally confined.

Locality in interactions is a strong requirement when the number of components in a system increases, or as the dimension of the distribution scale increases. In any case, tracking and controlling concurrent, autonomous, and autonomously initiated interactions is much more difficult than in object-based and component-based applications, even if these interactions are strictly local.

Examples

Control and manufacturing systems tend naturally to enforce local interactions. Each control component is delegated control of a portion of the environment, and its interactions with other components are usually limited to those that control neighboring portions of that environment, with which it typically is strictly coordinated.

In mobile computing, including applications for wearable systems and sensor networks, the very nature of wireless connections forces locality in interactions. Since wireless communication has limited range, a mobile computing component can directly interact ñ at a given time ñ only with a limited portion of the world [29].

Applications distributed in the Internet have to take into account the specific characteristics of the local administrative domain in which its components execute and have to interact, and components are usually allocated in Internet nodes so as to enforce as much as possible locality in interactions [8, 41].

2.5 A General Model

In sum, software systems increasingly follow the scheme of Figure 1. Software systems (dashed ellipses) are made up of autonomous components (black circles), interacting locally with each other and possibly with components of other systems. Some components may belong to several systems at different times. Systems and components are immersed in an environment, typically modeled as a set of environment partitions. Components in a system can sense and affect a local portion of the environment. Also, since the portions of the environment that two components may access may overlap, two components may interact indirectly via the environment.

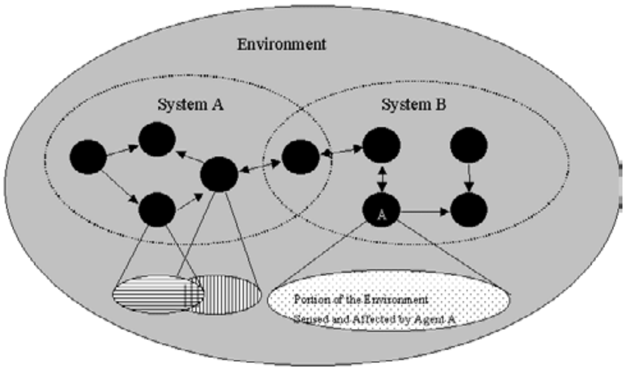


Fig. 1. The Scenario of Modern Software Systems

The scenario of Figure 1 is very different from component-based and object-based programming, and matches the prevailing model of agent-based computing [20]. Agents are situated software entities (they live in an environment) that execute autonomously (have local control of their actions) and interact with other agents. Local interactions are often promoted, although they may not be explicitly mentioned as part of the model. Despite this fact, most of the scientists working on agent-based computing still focus mostly on the AI aspects of the discipline, without noticing (or simply deliberately ignoring) that agents and agent-based computing have the potential to be a general model for today's computing systems and that different research communities face similar problems and are starting adopting similar modeling techniques. Similarly, outside the AI community, computer scientists often fail to recognize that they are already doing systems that can be assimilated to and modeled as agent-based systems.

The issues discussed in this paper can help clarify these relationships and the need for more interaction between different research communities, due to the strong similarities (we are tempted to say isomorphism) between the addressed problems.

3 Changing Our Attitudes

Traditionally, software systems are modeled from a mechanical stance, and engineered from a design stance. Computer scientists are both burdened and fascinated by the urge to define suitable formal theories of computation, to prove properties of software systems, and to provide a formal framework for engineering. Software engineers are used to analyzing the functionality that a system should exhibit in a top-down way, and to designing software architectures as reliable multi-component machines, capable of providing the required functionality efficiently and predictably.

In the future, scientists and engineers will have to design software systems to execute in a world where innumerable autonomous, embedded, and mobile software components are already executing and locally interacting with each other and with the environment. Such a scenario may require untraditional models and methodologies.

3.1 How Will Computer Science Change?

Modeling and handling systems with many components can be feasible if such components are not autonomous, i.e., are subject to a single flow of control. However, when the activities of these components are autonomous, it is hard, if not conceptually and computationally infeasible, to track them one by one and, so, to describe precisely the behavior of a system in terms of the behavior of its components. In addition, as several software systems are distributed and subject to decentralized control, or possibly embedded in some unreachable environment [19], tracking components and controlling their behavior is simply impossible. Such systems can only be described and modeled as a whole, in terms of macro-level observable characteristics, just as a chemist describes a gas in terms of macro properties like pressure and temperature.

This problem is exacerbated by the fact that components will interact with each other. Accordingly, the overall behavior of a system will emerge not only as the sum of the behavior of its components, but also as a result of their interactions. One may argue that since interactions tend to local (Subsection 2.4), this should not represent a big problem and, instead, it should be quite easy to control the effect of interactions. Unfortunately, the effect of local interactions can propagate globally and be very difficult to predict [31]. Even if one knows the initial status of the system very accurately, nonlinearities can amplify small deviations to become arbitrarily great.

As an additional problem, software systems will execute in an open and dynamic scenario, where new components can be added and removed at any time, and where some of these components can autonomously move from one part of the system to another. Thus, it is difficult to predict and control precisely not only the global dynamic behavior of the system, but also how such behavior can be influenced by such openness. In fact, it has been shown that the effects of interactions of autonomous active components strongly depends on the structure of the interaction graph [38]: strictly local interactions can produce a dynamic behavior that is completely different from the one emerging as soon as some form of non-locality in interactions is introduced, changing the structure of the interaction graph. Such problems can emerge in open systems, where the possibility for components to join and leave a system at any time dynamically changes the interaction graph and, possibly, its dynamic behavior.

Situatedness causes a similar problem. Modern thermodynamic and social sciences show that environmental forces can produce strange large scale dynamic behaviors in situated physical, biological, and social systems [31]. We can expect similar effects in situated software systems, because of the dynamics of the environment.

These problems will force computer scientists to change their attitudes dramatically in modeling complex software systems. Concurrency and interactivity already challenge the dream of dealing with traditionally formalizable systems [39], and it will become even more impractical in the future. Traditional formalisms can deal effectively with only small portions of large systems. The next challenge is to find alternative models or, more radically, to adopt a new scientific background for the study of software systems, enabling us to study, predict, and control the properties of a system and its dynamic behavior despite the inability to control its individual components.

Signals. Some signals of this trend are appearing in the research community.

Recent study and monitoring activities on the Web [4, 12] and on P2P networks [32] show that unpredictable and large-scale behaviors are already here, requiring new models and tools for their description. For instance, traditional Web caching policies are no longer effective when peculiar dynamic Web-access patterns emerge [12] and traditional reliability models fall short due to the specific emergent characteristics of Web and P2P networks [5, 32].

Some approaches to model and describe software systems in terms of thermodynamic systems have already been proposed [27, 28]. The ideas behind such research are twofold: to provide synthetic indicators capable of measuring how closely the system is approaching the desired behavior, and to provide tools to measure the influence of the environmental dynamics on the system. To some extent, a similar approach has been adopted in the area of massively parallel computing, where the need to measure specific global systems properties dynamically requires the introduction of synthetic indicators [11]. Similarly, it has been recognized that modeling large and dynamic (overlay) networks can only be faced by introducing macro properties rather than by direct representation of the network [4, 32].

One area that clearly shows such a trend is modern artificial intelligence. The concept of "rational" intelligence, as it should have emerged from a complex machine capable of manipulating facts and logic theories, is being abandoned. Now, the abstractions promoted by agent-based computing (matching the characteristics of today's software systems, Subsection 2.5) have shifted the emphasis to the concept of "intentional" intelligence, i.e., the capability of a component or of a system to behave autonomously so as to achieve a given goal. Organizational [42] and social science [24] are starting to influence research, as it is recognized that the behavior of a large scale software system can be assimilated more appropriately to a human organization aimed at reaching a global organizational goal, or to a society in which the overall global behavior derives from the self-interested intentional behavior of its individual members, than to a logical or mechanical system. Similarly, inspiration for computer scientists comes increasingly from the complex mechanisms of biological ecosystems, as well as the mindsets of biological sciences [15, 17].

We expect theories and models from complex dynamical systems, modern thermodynamics, biology, social science, and organizational science, will have to become more and more the sine-qua-non cultural background of the computer scientist.

3.2 How Will Software Engineering Change?

The change in the modeling and understanding of complex software systems will also impact how such systems are designed, maintained, and tested.

Today, software systems are designed to exhibit specific, predictable, and deterministic behavior at any level, from single units up to the whole system. However, in the presence of autonomous components situated in an open and dynamic environment, obtaining predictable behavior of a multi-component system in mechanical and deterministic terms is not feasible. The next challenge for the effective construction of large software systems, overcoming the impossibility of controlling the behavior of each of its single components and their interactions, is to build it so as to guarantee that the system as a whole will behave reasonably close to an ideal desired behavior despite the lack of exact knowledge about its micro-behavior. For instance, by adopting a "physical" attitude toward software design, a possible approach could be to build a system that, despite uncertainty on the initial conditions, is able to reach a given stable basin of attraction. By adopting a "teleological" attitude, one could build an ecosystem, or a society of components, able to behave in an intentional way, and robust enough to direct its global activities toward the achievement of the required goal.

The design of a system must also explicitly take into account the fact that the system will be immersed in an open and dynamic environment, and that the behavior of the system cannot be designed in isolation. Rather, the environment and its dynamics, as well as those software components that can enter and leave the systems, must become primary components of the design abstractions. A *defensive* design treats these factors as sources of uncertainty that can somehow be damaging to the global behavior of a system and that the system should be prepared to face. An *offensive* design considers openness and environmental dynamics as additional design dimensions to be exploited with the possibility of improving the behavior of the system [28].

Signals. Again, a few exemplars adopt this novel engineering perspective.

In the area of distributed operating systems management, policies for the management of distributed resources are already being designed in terms of autonomous components able to guarantee a global goal via local actions and local interactions, despite the dynamics of the environment [13]. The design of these policies is, in several cases, inspired by physical phenomena such as diffusion, which can re-establish global equilibrium in dynamic situations [11], despite the fact that such equilibrium will never be perfect but always locally perturbed.

Systems of ant colonies designed bottom-up can solve complex problems that resist traditional approaches, using very simple autonomous components interacting via a dynamic environment [26]. The idea is to mimic in software the behavior of insect colonies living in a dynamic world and able to solve, as a group and via the work sacrifice of a large number of worker insects (which translates in computational sacrifice to be paid), problems that have an interesting computational counterpart (i.e., sorting and routing). In that case, the environmental dynamics plays a primary role in design and in the emergence of specific useful behaviors.

Social phenomena like epidemics have inspired distributed information in dynamic networks of components, such as mobile ad-hoc networks [23], despite the inability of exactly controlling the information paths and structure of a network. The dynamics of the network is both a source of uncertainty and a useful property to guarantee that a message propagates to the whole network in a reasonable time. P2P information dissemination systems similarly exploit the dynamic properties of a spontaneously generated community network [32].

Both the dimension and the intrinsic dynamics of the network make it impossible to detect the structure of a network and of its components, challenging the whole concepts of information routing and information retrieval. In different areas (e.g., pervasive and mobile computing [3], P2P Internet computing [33, 35], middleware [10, 22], sensor networks [14]), there is a general understanding that the dynamics of networks require novel approaches to information retrieval and routing focused on content rather than paths. In other words, the basic idea is that the path to information sources/destinations should be dynamically found by relying on abstract overlay networks, dynamically and automatically reshaping themselves in response to system dynamics. Data or queries follow the shape of the overlay network just as a ball rolls down a surface. Whatever the final destination, the paths that emerge always proceed in a ìgoodî direction and eventually reach a point where the nodes in the network and the routed message are mutually appropriate.

Despite its intrinsic uncertainty, biological evolution is also likely to be a useful tool for software engineers. For instance, though cellular automata can perform complex computations, it is almost impossible to understand which rules must be imposed on cells and on their interaction to produce a system with required properties. An approach that has been successfully experienced is to make cellular automata rules evolve (e.g., via genetic algorithms) and eventually come up with specific rules leading to the desired global behavior of the automata, without anyone having directly designed such behaviors [34].

In addition to the change in the way software is designed, the new scenario will also dramatically impact the way software is tested, maintained, and evaluated.

For software systems conceived mechanically, testing mainly amounts to analyzing system state transitions to see if they correspond to those designed or if some of the components exhibit bad state transitions, i.e., bugs. Such work is very hard for large (even non-concurrent) systems, and may become impossible when autonomy and environmental dynamics produce a practically uncountable number of states. Thus, the testing criterion for a large software system will no longer be the absence of errors, but rather its ability to behave as needed as a whole, independently of the exact behavior of its components and of their initial conditions [18]. In an existing dynamic environment, where other systems are already executing and cannot be stopped, software cannot be simply tested and evaluated in terms of its capability to achieve the required goal. The test must also evaluate the effect of the environment on the software system and the effects of the software system on the environment. The better and more robust a system, the more reliably it can advance its goals independently of the dynamics of the environment, and the lower its impact on the surrounding environment and, thus, on other software systems. As a notable example, several approaches to content-based routing tend to evaluate their proposals in terms of how much the system can tolerate network dynamics by still exhibiting reasonably good behaviors, how fast the overlay network can re-organize in response to dynamic changes, and how the addition and removal of components impact network behavior [5].

Software maintenance will change too. Autonomy and openness mean that no software system will be stable, but to some extent needs maintenance continually due to changed conditions. When a large software system no longer behaves as needed (e.g., when external conditions require changes), update will no longer imply stopping the system, rebuilding it, and retesting it. Instead, it will imply intervening externally, by adding new components with different attitudes and by removing some of its existing components so as to change, as needed, the overall behavior of the system. The openness and situatedness of the system and the autonomy of its components can also make maintenance and updating smoother and less expensive, in that the system is already designed to tolerate and support dynamic changes in its structure.

3.3 Discussion

Humans increasingly rely on software systems for everyday activities, as well as for control of critical situations. One possible criticism of this approach is that such applications cannot be engineered in the way we have envisioned. Instead, one could insist that a software system exhibit a predictable and fully controllable behavior in each of its parts, and that the duty of a software engineer is to produce such reliable systems.

A few MAS researchers are pioneering such new perspective. But many others, aware that emergent behavior is likely to appear in open systems of autonomous components, consider all such behavior as undesirable. For instance, pessimists foresee the death of P2P approaches because of their unreliability and the impossibility of proper modeling. In our opinion, this is not the correct approach. Constraining the behavior of a highly interactive system may sacrifice most of its computational power and may require much more waste of resources to obtain the same functionalities. The engineering work needed to make a system fully controllable may be greater than the work needed to produce emergent behavior that is useful, reliable, and stable. Finally, constraining ìby designî a software system may simply make it lose the properties needed to support openness and to tolerate environmental dynamics.

In any case, we are aware that, for our vision on software engineering to become practical and used, much theoretical, methodological, and experimental work is required, paving the way for a suitable set of conceptual tools and frameworks to be exploited by the engineers of next generation software systems.

4 Conclusions

Modern software systems, in different application areas, exhibit characteristics that make them very different from the software systems to which we, as scientists and engineers, are accustomed. These characteristics are likely to impact dramatically the very way software systems will be modeled and engineered, leading to a true revolution in computer science and software engineering [9, 21]. In fact, we will be required to change from our traditional "design" attitude, implying a mechanical perspective, to an "intentional" attitude, requiring physical, biological, and teleological perspectives. Despite the opposing forces and the difficulties inherent in any revolutionary phase, including the need of restructuring our cultural background, this revolution will open the door to new interesting research and engineering challenges in which, hopefully, the multi-agent research community will hold a leading position.

References

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, ìAmorphous Computingî, Communications of the ACM, 43(5), May 2000.

[2] G. D. Abowd, E. D. Mynatt, ìCharting Past, Present and Future Research in Ubiquitous Computingî, ACM Transactions on Computer-Human Interaction, 7(1):29-58, March 2000.

[3] W. Adjie-Winoto, E. Schwartz, H. Balakrishna, J. Lilley, ìThe Design and Implementation of an Intentional Naming Systemsî, 17th ACM Symposium on Operating Systems Principles (SOSP '99), ACM, 1999.

[4] R. Albert, H. Jeong, A. Barabasi, ìDiameter of the World Wide Webî, Nature, 401:130-131, 9 Sept. 1999.

[5] R. Albert, H. Jeong, A. Barabasi, ìError and Attack Tolerance of Complex Networksî, Nature, 406:378-382, 27 July 2000.

[6] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, J. Jetcheva, ìA Performance Comparison of Multi-hop Wireless Ad-Hoc Network Routing Protocolsî, 3rd ACM/IEEE Conference on Mobile Computing and Networking, October 1998.

[7] S. Bussmann, ìSelf-Organizing Manufacturing Control: an Industrial Application of Agent-Technologyî, 4th IEEE International Conference on Multiagent Systems, Boston (MA), July 2000, pp. 87-94.

[8] G. Cabri, L. Leonardi, F. Zambonelli, ìEngineering Mobile Agent Applications via Context-Dependent Coordinationî, IEEE Transactions on Software Engineering, 11(9), Nov. 2002.

[9] F. Capra, The Web of Life: The New Understanding of Living Systems, Doubleday, Oct. 1997.

[10] A. Carzaniga, D. S. Rosenblum, A. L Wolf, ìDesign and Evaluation of a Wide-Area Event-Notification Serviceî, ACM Transactions on Computer Systems, 19(3):332-383, Aug. 2001.

[11] A. Corradi, L. Leonardi, F. Zambonelli, ìDiffusive Load Balancing Policies for Dynamic Applicationsî, IEEE Concurrency, 7(1):22-31, 1999.

[12] M. Crovella, A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Causes" ACM Sigmetrics, pp. 160-169, 1996.

[13] G. Cybenko, ìDynamic Load Balancing for Distributed Memory Multiprocessorsî, Journal of Parallel & Distributed Computing, 7(2), Feb. 1989.

[14] D. Estrin at al., ìConnecting the Physical World with Pervasive Networksî, IEEE Pervasive Computing, 1(1):59-69, January 2002.

[15] R. Gustavsson, M. Fredriksson, ìCoordination and Control in Computational Ecosystems: A Vision of the Future, in Coordination of Internet Agents, A. Omicini et al. (Eds.), Springer Verlag, pp. 443-469, 2001.

[16] A. Howard, M. J. Mataric, ìCover Me! A Self-Deployment Algorithm for Mobile Sensor Networksî, International Conference on Robotics and Automation, 2002, to appear.

[17] B. A. Huberman, T. Hogg, "The Emergence of Computational Ecologies", in Lectures in Complex Systems, Addison-Wesley, 1993.

[18] M. Huhns, "Interaction-Oriented Programming", 1st International Workshop on Agent-Oriented Software Engineering, LNCS No. 1957, Jan. 2001.

[19] C. Intanagonwiway, R. Govindam, D. Estrin, ìDirected Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networksî, 5th ACM/IEEE Conference on Mobile Computing and Networking, Boston (MA), Aug. 2000, pp. 56-67.

[20] N. R. Jennings, "An Agent-Based Approach for Building Complex Software System", Communications of the ACM, 44(4):35:41, 2001.

[21] T. Kuhn, The Structure of Scientific Revolutions, University of Chicago Press, 3rd Edition, Nov. 1996.

[22] M. Mamei, L. Leonardi, F. Zambonelli, ìA Physically Grounded Approach to Coordinate Movements in a Teamî, 1st International Workshop on Mobile Teamwork at ICDCS, IEEE CS Press, July 2002.

[23] W. G. Mitchener, A. Vahdat, ìEpidemic Routing for Partially Connected Ad-Hoc Networksî, Duke Technical Report, No. CS-2000-06, July 2000.

[24] Y. Moses, M. Tenneholtz, ìArtificial Social Systemsî, Computers and Artificial Intelligence, 14(3):533-562, 1995.

[25] P. Noriega, C. Sierra, J. A. Rodriguez, ìThe Fishmarket Project. Reflections on Agent-mediated institutions for trustworthy E-Commerceî, 1st Workshop on Agent Mediated Electronic Commerce (AMEC-98), 1998.

[26] V. Parunak, ìGo to the Ant: Engineering Principles from Natural Agent Systemsî, Annals of Operations Research, 75:69-101, 1997.

[27] V. Parunak, S. Bruekner, "Entropy and Self-Organization in Agent Systems", 5th International Conference on Autonomous Agents, ACM Press, May 2001.

[28] V. Parunak, S. Bruekner, J. Sauter, "ERIMís Approach to Fine-Grained Agents", NASA/JPL Workshop on Radical Agent Concepts, Greenbelt (MD), Jan. 2002.

[29] G. P. Picco, A.M. Murphy, G.-C. Roman, ìSoftware Engineering for Mobility: A Roadmapî, in The Future of Software Engineering, A. Finkelstein (Ed.), ACM Press, pp. 241-258, 2000.

[30] N.B. Priyantha, A.K.L. Miu, H. Balakrishnan, S. Teller, ìThe Cricket Compass for Context-aware Mobile Applicationsî, 6th ACM/IEEE Conference on Mobile Computing and Networking, Rome (I), July, 2001.

[31] I. Prigogine, I. Steingers, The End of Certainty: Time, Chaos, and the New Laws of Nature, Free Press, 1997.

[32] M. Ripeani, A. Iamnitchi, I. Foster, ìMapping the Gnutella Networkî, IEEE Internet Computing, 6(1):50-57, Jan.-Feb. 2002.

[33] A. Rowstron, P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems", 18th IFIP/ACM Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, Nov. 2001.

[34] M. Sipper. ìThe Emergence of Cellular Computingî, IEEE Computer, 37(7):18-26, July 1999.

[35] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, ìChord: A Scalable Peer-to-Peer Lookup Service for Internet Applicationsî, ACM SIGCOMM Conference 2001, San Diego (CA), Aug. 2001.

[36] R. Tolksdorf, "Coordinating Work on the Web with Workspaces", 9th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, Gaithersburg (MA), IEEE CS Press, June 2000.

[37] D. Tennenhouse, "Proactive Computing", Communications of the ACM, 43(5):43-50, May 2000.

[38] D. Watts, Small Worlds : The Dynamics of Networks between Order and Randomness, Princeton University Press (Princeton, NJ), 1999.

[39] P. Wegner. "Why Interaction is More Powerful than Algorithms", Communications of the ACM, 1997.

[40] M. Weiser, ìHot Topics: Ubiquitous Computingî, IEEE Computer, 26(10), October 1993.

[41] J. White, ìMobile Agentsî, in Software Agents, J. Bradshaw (Ed.), AAAI Press, Menlo Park (CA), pp. 437-472, 1997.

[42] F. Zambonelli, N. R. Jennings, M. J. Wooldridge, ìOrganizational Rules as an Abstractions for the Analysis and Design of Multi-agent Systems, International Journal of Knowledge and Software Engineering, 11(4), April. 2001.

[43] W. G. Zitchener, A. Vahdat, ìEpidemic Routing for Partially Connected Ad-Hoc Networksî, Duke University Technical Report, No. CS-2000-06, July 2000.

Rationality, Autonomy and Coordination: The Sunk Costs Perspective

Matteo Bonifacio, Paolo Bouquet, Roberta Ferrario, and Diego Ponte

University of Trento
Via Belenzani, 12 – I-38100 Trento, Italy
{bonifacio,ponte}@itc.it, bouquet@dit.unitn.it, and ferrix@cs.unitn.it

Abstract. Our thesis is that an agent[1] is autonomous only if he is capable, within a non predictable environment, to balance two forms of rationality: one that, given goals and preferences, enables him to select the best course of action (means-ends), the other, given current achievements and capabilities, enables him to adapt preferences and future goals. We will propose the basic elements of an economic model that should explain how and why this balance is achieved: in particular we underline that an agent's capabilities can often be considered as partially sunk investments. This leads an agent, while choosing, to consider not just the value generated by the achievement of a goal, but also the lost value generated by the non use of existing capabilities. We will propose that, under particular conditions, an agent, in order to be rational, could be led to perform a rationalization process of justification that changes preferences and goals according to his current state and available capabilities. Moreover, we propose that such a behaviour could offer a new perspective on the notion of autonomy and on the social process of coordination.

1 Rationality in Traditional Theories of Choice

Traditional theories of choice are based upon the paradigm that choosing implies deciding the best course of action in order to achieve a goal [31]. Goals are generally considered as given or, at least, they are selected through an exogenous preference function which assigns an absolute value to each possible state of the world [29]. Potential goals, once ordered according to preferences, are selected by comparing each absolute value with the cost of its achievement. In particular, the agent will commit to the goal that maximizes the difference between the absolute benefit of the goal and the cost of using the capabilities that are needed. This means-ends paradigm subtends a type of rationality that March defines as anticipatory, causative, consequential, since an agent anticipates the consequences of his actions through a knowledge of cause-effect relationships [9, 24]. Here, as underlined by Castelfranchi, autonomy is viewed in the restrictive sense of *executive autonomy*: the only discretionality the agent possesses is about the

[1] In this paper we do not intentionally draw any distinction between artificial and human agents, but we rather discuss the concept of agent in general.

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 29–38, 2003.

way in which a goal is to be achieved and not about which kind of goal should be preferable; in this sense, even if an agent selects a goal, he is unable to direct the criteria of the selection. The interest of the agent is always reconducible to the one of the designer and, as Steels concludes referring to Artificial Agents, “AI systems built using the classical approach are not autonomous, although they are automatic . . . these systems can never step outside the boundaries of what was foreseen by the designer because they cannot change their own behaviour in a fundamental way.” [34]. Sometimes, as we will propose, autonomy and rationality lie in the possibility to change our mind on what is good and what is bad on the basis of current experience; basically, this is equivalent to the possibility to decide not just how to achieve a goal, but rather which goal is to achieve and, moreover, which is preferable.

2 Another Perspective on Rationality: Ex-post Rationalization

Another way to look at rationality, that March defines as ex-post rationality or rationalization offers an opposite perspective on decision making [25]. At the extreme, it envisions an agent as somebody who first acts and then justifies his actions defining appropriate goals and preferences in order to be consistent to his current achievements. More realistically, it presents an agent not as somebody who is only able to be rational in the sense of setting appropriate courses of action, but also in the sense of changing his mind about what is preferable when planned achievements become unrealistic [28]. Such an agent is able to learn not just in terms of finding better ways to achieve a goal but also in terms of finding goals that are more appropriate to his capabilities. As we will see afterwards, in an environment characterized by a non predictable evolution, an agent who has a partial and perspective view of the world [3, 17] will often come to situations in which ex-post rationalization is more rational than setting a plan for the achievement of given goals [29]. We will propose that this process hides an economic principle of reuse and conservation that could lead an agent to try to fit the world, rather than pretending the world to be appropriate to him.

Moreover, if non predictability is the main reason to be rational and autonomous in the sense just stated, ex post rationalization is also an opportunity for the agent to be like this. In particular, whenever an environment is ambiguous and undefined, equally ambiguous and undefined is the definition of what is good and what is bad. More simply, we often describe a situation as good or bad not because it is so in itself, but rather because of our interpretation and our convenience; as commonly said, it is a question of perspective [11]. Here “rationalization” appears as an opportunity, since it can hide a powerful tool to learn from experience, which produces as outcome the possibility of seeing the world from different perspectives. As underlined by [29], this view represents decisions as constructive interpretations, since they “are often reached by focusing on reasons that justify the selection of one option over another. Different frames, contexts, and elicitation procedures highlight different aspects of the options and bring forth different reasons and considerations that influence decisions". More simply, thanks to rationalization, an agent can understand that, under a different point of view, a mistake or an unlucky event could become an opportunity to learn. In this sense the "value" of a goal appears to be a choice rather than an evidence [24].

3 The Rationale of Rationalization: Sunk Costs, Economies of Scale and Irreversibility

In this work we propose that ex-post rationalization can find a rational justification in the sunk cost effect which derives from the co-occurrence of two conditions that could characterize an agent's capability: economies of scale and irreversibility. To start, an agent has a set of means that we can view both as capabilities when used to perform actions and as resources when used to develop or acquire new capabilities. Under this perspective, at a given moment, an agent's set of capabilities can be interpreted as the result of an investment of resources. Traditional theories of choice assume that the value of an investment (that generates an agent's capability) is realized in its use and, in general, when calculating the costs of a decision, we consider just those that refer to currently used capabilities [22].

The observation of the process of decision shows something different. In particular:

- when calculating the cost of a decision, we consider not just the cost of those capabilities that we use, but also the cost generated by the non use of some other one. This is because the generation of a capability implies the sustenance of some fixed costs that can be amortized through its repeated use. In fact, in presence of fixed costs, reuse implies a decrease in the unitary cost of each reutilization. In economic theory, this effect is called the economies of reuse effect. Consequently, not using a resource implies a loss of value generated by the lost opportunity of a cost saving. In other words, each time we use a capability we exploit its economies of reuse effect and at the same time we loose the correspondent effect of those we do not use.
- each capability, when considered as a resource that can be used to acquire another capability, displays a rate of irreversibility. This is because when trying to transform it into another, we can sustain a loss in value if the resource is difficult to manipulate or if it is difficult to find a buyer on the market. In general, if a resource is totally reversible (for example currency), it can be sold on the market and the economies of reuse effect has only marginal impact on the decision of the agent. On the other hand, if totally irreversible, the resource will completely display its economies of reuse effect; if this is not used, its owner will suffer the loss of potential cost saving [12].

In all these cases, a resource which is characterized both by economies of reuse and a rate of irreversibility is considered a sunk cost and it generates the effect that "paying for the right to use a good will increase the rate at which the good will be utilized ceteris paribus" [36]. This hypothesis will be referred to as the sunk cost effect [1]. The main stream in sunk cost studies underlines the irrational and psychological nature of such behavior, referring to:

– the need to justify prior choices [6];
– avoidance of waste [2];
– the tendency to be risk seeking in the light of previous losses [16].

Some authors have remarked that this tendency can lead to irrational behaviours such as the "irrational escalation" [33] [32], whereby social agents could irrationally justify the current failure in order to explain past choices and "save their face" [7]. In fact, this tendency (the sunk cost consideration) contradicts a basic principle in economics that past costs and benefits should be irrelevant to current decisions [15]. In [22] Johnstone writes: "For decision-making purposes, sunk costs are strictly irrelevant. This is a law of economic logic justified by the argument that because no action (current or future) can avert or reduce a sunk cost, no sunk cost can be attributed to or have any relevance to current or future action. It is evident, however, that for many of us, the edict that sunk costs be ignored is hard to accept, if not as a matter of logic then at least in application"[2].

On the other hand, consistently to the observation raised by Johnstone, we underline that a different school of thought, following the so called "Decision Dilemma Theory" [4], does not consider the sunk cost effected behaviour as irrational. The basic idea is that, in complex environments where there is ambiguous feedback (feedback on the basis of which multiple interpretations can be constructed[3]) on actions and outcomes [37], it is often impossible to objectively determine whether further investments will bring about success or failure. As Bowen says, escalations are not necessarily erroneous because:

> "...if information is equivocal when a decision is necessary, a recommitment of resources to a course of action simply may offer an additional opportunity to permit a strategy to work, to demonstrate its inability to produce desired results, or to allow for the collection of additional data and the passage of time which might promote an increased understanding of the situation." [4]

In particular, there are two important theories which give an alternative explanation of the sunk cost effect: the "reinforcement theory" perspective [19] and the "attribution theory" perspective [26]. Both theories hold that individuals learn from similar situations whether to continue to invest [21]. Moreover, decision makers learn from past feedback how to move forward in the context of the decision.

[2] See also [30, 35].

[3] "It appears that in escalation situations, equivocality encompasses both the variability of feedback data described by uncertainty and especially contextual factors which may influence interpretations." [21]

Reinforcement history scholars point out that individuals can learn how a reward can follow a number of nonrewards. This is because people tend to build chains with past outcomes in order to make predictions and, as O'Flaherty and Komaki demonstrate, this kind of analysis generates interpretations that are consistent with predictions derivable from Bayesian updating [20, 27]. In this sense, in ambiguous environments more than one reasonable history can be constructed with respect to each outcome and this influences further decisions.

Attribution theory scholars think that delaying exit decisions under equivocal conditions is useful for decision makers, in order to gather more information about the situation. On the other hand, when facing ambiguous situations, as people try to gather further information, past equivocal feedback may influence the subsequent feedback, making it more equivocal [21].

In general, when a situation doesn't have a definite outcome, a particular decision or series of decisions cannot necessarily be considered as erroneous (affected from sunk costs). Such decisions, instead of being viewed as instances of escalation of commitment, are rather to be considered a "retrospective interpretation of fallacy" done by the external observer [5], because "...the interpretation were predicated on post-hoc analyses that focused on the primacy of hindsight over foresight." [21]. Differently said, in ambiguous situations, a behavior can be judged as irrational only ex-post (after the decision has produced its consequences) and not ex-ante (before the decision has been made). In such cases sunk cost driven strategies may be considered as normatively rational.

Now the point is that, in a non predictable environment, while pursuing a goal, an agent can be led to develop and acquire capabilities that, to some extent, have no use in order to achieve his current goal. This probably happens in cases of very turbulent environments and the phenomenon is enhanced when the agent is in an advanced stage of his life or is particularly experienced in the domain of the decision he is facing; the former circumstance leads the agent to unforeseen situations and to the generation of redundant capabilities; the latter to the growing accumulation of sunk investments and costs, as those that are more reversible were probably used during the earlier stages of the agent's life or experience. As a consequence, since these resources display both an economy of reuse and a rate of irreversibility, they should generate an incentive to reuse in order to exploit the value of past investments. Here, an agent that is facing an ambiguous decision context should display, in order to be rational, a sunk cost affected decision making process. In fact, while deciding, an agent has to consider not just the currently sustained costs, but also those losses generated by the non use of sunk investments.

4 Generating Preferences and Goals

In this section we will give evidence on how a decision making process that considers sunk costs can lead to an ex-post rationalization whereby an agent manipulates preferences in order to justify his current state. As we will propose, through this endogenous process of preferences formation, an agent becomes able to select and pursue goals which are not foreseeable a-priori.

4.1 Generating Preferences

As [16] suggests, the “commitment to a current course of action is a function of the comparison between the perceived utility of continuing with the action and the perceived utility of withdrawal and/or changing the action”. If sunk costs are taken into consideration, decisions generating investments influence future choices that are constrained by the need to preserve past investments. As a consequence, in determining which option is preferable in a decision, an agent can face a situation in which the cost of changing his mind about what is preferable is lower than the cost of going on in the pursuit of his intentions. In particular, in the decision function the weight of sunk costs overcomes the one of current opportunities.

Such a situation offers the agent the opportunity to do something qualitatively different: instead of reasoning on how to pursue an unrealistic goal, he could realistically consider his current state as appropriate to his capabilities. In this case, a current unexpected situation could be viewed by the agent as a proper ex-post goal, and remaining where he is could be more rational than moving. As argued before, instead of reasoning about means necessary to achieve ends which were shown to be irrational, he rationalizes his current state as an end which is appropriate to his means. Under this perspective, the sunk cost effect is an attempt to demonstrate the rationality of behaviours that are otherwise not explained and thus labelled as “irrational” by traditional theories of rationality. Consistently to the view proposed by March, this attitude leads to a process of retrospective self-justification that implies a change in preferences [23]. In fact, in order to be consistent with his history, an agent who rationalizes his current state needs to change his preferences accordingly. As a matter of fact, in order to justify the (even ex-post) adoption of a goal, a rational agent needs to express such a goal as desirable. If not, the agent would display the inconsistent behaviour of choosing a goal which is not desirable. This necessity leads the agent to invert his reasoning process on preferences which are turned from fixed tools used to select goals to variable matters that are adapted to (now fixed) current achievements. As clearly stated by [13], “When people realize they are in situations that they have never considered before, they do not judge themselves to be irrational. Instead, they simply try to decide what beliefs and preferences to adopt (if any)”. Said differently, it is rational for the agent to perform a counterfactual process [14] [18] that could be expressed by a sentence such as: “What should I have preferred in order to be satisfied with the state of the world I am currently in?” or “What should I have preferred in order to desire to reach a goal that is consistent with the current state of the world?”.

4.2 Generating Goals

As anticipated, through this counterfactual process, the agent asks himself which goal he should have been committed to in order to be, given his resources/capabilities, satisfied with what he currently is. In a particular sense, such a process represents the first attempt for an agent to endogenously generate a goal; the goal is the already achieved current state that, only ex-post, can be viewed as a goal. That is to say, we propose that the first manifestation of *goal autonomy* is the rationalization of an unexpected and undesired current state, turned into a desired one. Since rationalization is exactly driven by sunk costs, this first goal, by definition, will display the peculiarity of exploiting the value of current sunk investments. We underline that this original process of goal and preferences creation is not an abstract process of imagining new possible worlds and preferences but a concrete exercise that uses the presence of a goal (the current state) as a tool to derive a proper set of preferences.

At a first sight, the behaviour of this agent could appear to be intrinsically conservative. Once the weight of past investments overcomes the weight of immediate value, the agent stops where he is, due to the retrospective justification of his state as a desired one. Even if we think that, in time, conservative behaviours are an underlying tendency of the agent, we propose that, within this tendency, new deliberative behaviours can emerge. Here we give just an example of how new goals can emerge, derived by the idea of Castelfranchi [10] of social adoption. In fact, given the new set of preferences, the agent is now able to assign new "values" to every state of the world that is accessible to his knowledge. In this way an agent is able to reorder the states of the world on his new preference scale and set new goals. On the other hand, he has now changed his beliefs on means-ends relations and on how a particular set of capabilities (the ones used to reach the current state) can be used in order to achieve a goal. In particular, he changes his beliefs on what is preferable and on which means are needed in order to get to a particular goal (the current state) [13]. For example, we might argue that the agent, observing other agent's situations, discovers that another state of the world displays a net benefit (considering the new preferences and existing sunk costs) which is higher than the one of preserving the current state. Now he will adopt the new possible state as the new goal. Again, as above, this process can lead to the acquisition of new resources and to the possibility that, on the path to the goal, the agent happens again to be in unexpected states of the world that might influence, through the evaluation of sunk investments, his preferences.

5 Conclusions: Autonomy and Coordination

This conclusion leads us to some considerations on the notion of autonomy. We agree with the one proposed by Castelfranchi [8] whereby an agent is autonomous if he is able to choose goals on the basis of a personal interest. Here we underline the need that such interest is an endogenous production of the agent rather than something exogenously given by a designer (in the case of artificial agents) or by another human or metaphysical entity (in the case of human agents). Moreover, Castelfranchi remarks that the definition of autonomy currently used in artificial agents literature is referable to the weaker notion of *executive autonomy* (as opposed to *goal autonomy*): an agent is autonomous if he is able to choose among alternative courses of action. As he underlines, this kind of autonomy could resolve both in a type of slavery (from some external utility function) and in a form of irrationality (pursuing some other's interest when this is conflicting with our own is irrational). We strongly believe as Castelfranchi in the idea that an agent, if not goal autonomous, is not autonomous at all and, moreover, potentially irrational. Now the question becomes how such a type of autonomy can emerge in order to design, if possible, agents that can display *goal autonomy* through the generation of endogenous preferences and the consequent adoption of non a-priori foreseeable goals. In this work, we sketch the lines of a model that could give an answer. In particular our thesis is that an agent, in order to be rational, endogenously develops preferences and goals that are consistent to his "emerging" interest. This interest is the consequence of an unforeseen evolution of his life that led to the generation of sunk costs that need to be considered in decision making. Such an evolution, assuming a non predictable environment, leads to the autonomous formation of preferences that are not foreseeable a-priori, and that are the rational consequence of an economic principle of reuse. Through preference formation, new goals become desirable while old ones are abandoned. In this sense we say that the agent, at a certain stage of his life, in order to be rational, needs to become autonomous (and form new preferences).

One last point addresses the way in which this approach could be used to interpret some fundamental aspects of an agent's sociality, in particular those aspects that involve coordination with other agents. Specifically, if we consider coordination efforts as investments that display a sunk cost effect, we can explain the persistency of social relations. We refer to the observation that social relations among social agents are less prone to opportunism than what is predicted by traditional utilitaristic theories. In fact, whenever the current value of a relation is lower than the cost of keeping it, an agent should break such relation. As a matter of fact, social relations seem to be more persistent than this. A way to interpret such persistency without recurring to exogenous factors (such as social norms) [10], is provided by the perspective of sunk costs. Here a social relationship is viewed as a resource and capability that displays an economies of reuse effect (as a consequence of the initial investment in creating the relation) and a rate of irreversibility (since a social relationship cannot always be transformed into another). As a consequence, an agent, in order to achieve his goal, will tend to reuse and justify current established relations before creating new ones.

References

[1] H. R. Arkes and P. Ayton. The Sunk Cost and Concorde Effect: are Humans Less Rational Than Lower Animals? *Psychological Bulletin*, 125, 1999. 32

[2] H. R. Arkes and C. Blumer. The psychology of sunk cost. *Organizational Behavior and Human Performance*, 35:129–140, 1985. 32

[3] M. Benerecetti, P. Bouquet, and C. Ghidini. Contextual reasoning distilled. *Journal of Theoretical and Experimental Artificial Intelligence*, 12(3):279–305, 2000. 30

[4] M. G. Bowen. The escalation phenomenon reconsidered: Decision dilemmas or decision errors? *Academy of Management Review*, 12:52–66, 1987. 32

[5] M. G. Bowen and F. C. Power. The moral manager: Communicative ethics and the exxon valdez disaster. *Business Ethics Quarterly*, 3:97–115, 1993. 33

[6] J. Brockner. The escalation of commitment to a failing course of action: towards theoretical progress. *Academy of Management Review*, 17:39–61, 1992. 32

[7] J. Brockner, J. Z. Rubin, and E. Lang. Face-saving and entrapment. *Journal of Experimental Social Psychology*, 17:68–79, 1981. 32

[8] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages*, pages 56–70. Springer-Verlag: Heidelberg, Germany, 1995. 35

[9] M. D. Cohen, J. G. March, and J. P. Olsen. A garbage can model of organizational choice. *Administrative Science Quartely*, 17:1–25, 1972. 29

[10] R. Conte and C. Castelfranchi. *Cognitive and social Action*. UCL Press, 1995. 35, 36

[11] R. L. Daft and K. E. Weick. Toward a model of organizations as interpretation systems. *Academy of Management Review*, 9(2):284–295, 1984. 30

[12] B. di Bernardo and E. Rullani. *Il management e le macchine*. il Mulino, 1990. 31

[13] J. Doyle. Rationality and its roles in reasoning. *Computational Intelligence*, 8(2):376–409, 1992. 34, 35

[14] R. Ferrario. Counterfactual reasoning. In V. Akman, P. Bouquet, R. Thomason, and R. A. Young, editors, *Modeling and Using Context*, LNAI 2116, pages 170–183, Dundee, UK, July 2001. Springer Verlag. 34

[15] R. H. Frank. *Microeconomics and Behavior*. McGraw Hill: New York, 1994. 32

[16] H. Garland and S. Newport. Effects of Absolute and Relative Sunk Cost on the Decision to Persist with a Course of Action. *Organizational Behavior and Human Decision Processes*, 48:55–69, 1991. 32, 34

[17] C. Ghidini and F. Giunchiglia. Local models semantics, or contextual reasoning = locality + compatibility. *AI*, 127(2):221–259, April 2001. 30

[18] M. L. Ginsberg. Counterfactuals. *AI*, 30(1):35–79, 1986. 34

[19] S. M. Goltz. A sequential learning analysis of decisions in organizations to escalate resources investments despite continuing costs or losses. *Journal of Applied Behavior Analysis*, 25:561–574, 1992. 32

[20] S. M. Goltz. Examining the joint roles of responsibility and reinforcement history in recommitment. *Decision Sciences*, 24:977–944, 1993. 33

[21] D. H. Hantula and J. L. DeNicolis Bragger. The effects of feedback equivocality on escalation of commitment: An empirical investigation of decision dilemma theory. *Journal of Applied Social Psychology*, 29:424–444, 1999. 32, 33

[22] D. Johnstone. The reverse sunk cost effect and explanation: rational and irrational. http://www.departments.bucknell.edu/management/apfa/papers/17Johnstone.pdf, 2000. 31, 32

[23] J. G. March. *Decisions and Organizations*. T. J. Press Ltd., 1988. 34

[24] J. G. March. How decisions happen in organizations. *Human Computer Interaction*, 6:95–117, 1991. 29, 31

[25] J. G. March. *A primer on decision making : how decisions happen*. The Free Press, 1994. 30

[26] B. E. McCain. Continuing investment under conditions of failure: A laboratory study of the limits to escalation. *Journal of Applied Psychology*, 71:280–284, 1986. 32

[27] B. O'Flaherty and J. L. Komaki. Going beyond with bayesian updating. *Journal of Applied Behavior Analysis*, 25:590–612, 1992. 33

[28] J. W. Payne, R. Bettman, and R. J. Johnson. Behavioral decision research: a Constructive Processing Perspective. *Annual Review of Psychology*, 4:87–131, 1992. 30

[29] E. Shafir, I. Simonson, and A. Tversky. Reason-based choice. *Cognition*, 49, 1993. 29, 30

[30] M. Shefrin. *Beyond greed and fear: understanding behavioral finance and the psychology of investing*. Harvard Business School Press, 2000. 32

[31] H. A. Simon. *Reason in human affairs*. Stanford University Press, 1983. 29

[32] B. Staw. Knee-deep in the big muddy: a study of escalating commitment to a chosen course of action. *Organizational Behaviour and Human Performance*, 16:27–44, 1976. 32

[33] B. Staw and J. Ross. Understanding behavior in escalation situations. *Science*, 246:216–220, 1989. 32

[34] L. Steels. When are robots intelligent autonomous agents? *Journal of Robotics and Autonomous Systems*, 15:3–9, 1995. 30

[35] R. Thaler. Toward a theory of consumer choice. *Journal of Economic Behaviour and organization*, 1:39–60, 1980. 32

[36] R. Thaler. *Quasi rational economics*. Russel Sage foundation, 1994. 32

[37] E. K. Weick. *Sensemaking in Organizations*. Sage Publications, Inc., 1995. 32

A Normative and Intentional Agent Model for Organisation Modelling

Joaquim Filipe

Escola Superior de Tecnologia do Instituto PolitÈcnico de Set˙bal
Rua Vale de Chaves, Estefanilha, 2910 Set˙bal, Portugal
JFilipe@est.ips.pt

Abstract. This paper proposes a new agent model, including normative aspects based on studies from social psychology. The three main model components are the Epistemic, Deontic and Axiologic components. This model structure facilitates the representation of normative organisational knowledge and processes as well as the relations between individual and social interests. Instead of manipulating the notions of belief, intention, desire, commitment, goal and obligation, the only primitives needed are belief and generalized goal.

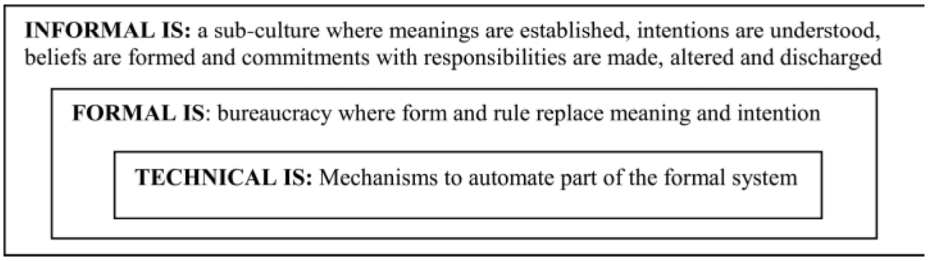
1 Introduction

Organisations are multi-agent systems, eventually including both human and artificial agents. Organisations are also seen as multilayered Information Systems (IS) themselves, including an informal subsystem, a formal subsystem and a technical system as shown in figure 1, denoting the organizational semiotics perspective on information systems.

We aim at improving the technical subsystem within the constraints defined by the other two. Organisational information systems are inherently distributed, nowadays, thus communication and coordination are major problems in this kind of information systems. Perhaps motivated by the difficult problems there is currently a strong interest on this area, which is an active research field for several disciplines, including Distributed Artificial Intelligence (DAI), Organisational Semiotics and the Language-Action Perspective, among others. Our approach integrates elements from these three perspectives.

The Epistemic-Deontic-Axiologic (EDA) designation refers to the three main components of the agent structure described in this paper. Here we propose an agent model which, contrary to most DAI proposals, not only accounts for intentionality but is also prepared for social interaction in a multi-agent setting. Existing agent models emphasise an intentional notion of agency ñ the supposition that agents should be understood primarily in terms of mental concepts such as beliefs, desires and intentions. The BDI model (Rao and Georgeff, 1991) is a paradigm of this kind of agent, although some attempts to include social aspects have been made (Dignum et al., 2000).

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 39-52, 2003.



INFORMAL IS: a sub-culture where meanings are established, intentions are understood, beliefs are formed and commitments with responsibilities are made, altered and discharged

FORMAL IS: bureaucracy where form and rule replace meaning and intention

TECHNICAL IS: Mechanisms to automate part of the formal system

Fig. 1. Three main layers of the real information system (Stamper 1996)

We claim that the cognitive notions that constitute the basis of intentional models show only one face of the coin, the other one being social agency. It is required that an adequate agent model, able to function in a multi-agent setting, should emphasise social agency and co-ordination, within a semiotics framework. Our approach focuses on organisational agents who participate in organisational processes involving the creation and exchange of signs, *i.e.* knowledge sharing, in a way that is inherently public, thus depending on the agentís social context, *i.e.* its information field (Stamper, 1996), to support co-ordinated behaviour. An information field is composed by all the agents and objects involved in a certain activity and sharing a common ontology.

A realistic social model must be normative: both human agents and correctly designed artificial agents ought to comply with the different kinds of norms that define the information field where they operate, although exceptions may occur. Private representations of this shared, normative, knowledge are translations of the public knowledge into specific agent mental structures. When designing an artificial agent, the designer must adopt a private knowledge representation paradigm to set up the agentís internal knowledge in such a way that it fits the normative shared ontology.

We postulate that norms are the basic building blocks upon which it is possible to build co-ordination among organised entities and co-ordinated actions are the crux of organised behaviour. We claim that although organised activity is possible either with or without communication, it is not possible without a shared set of norms. Therefore, these socially shared norms define an information field that is a necessary condition for heterogeneous multi-agent co-ordination including both artificial agents and humans.

2 The Normative Structure of the EDA Model

Norms are typically social phenomena. This is not only because they stem from the existence of some community but also because norms are multi-agent objects (Conte and Castelfranchi, 1995):

- They concern more than one individual (the information field involves a community)
- They express someoneís desires and assign tasks to someone else

- Norms may be regarded from different points of view, deriving from their social circulation: in each case a norm plays a different cognitive role, be it a simple belief, a goal, a value, or something else.

Social psychology provides a well-known classification of norms, partitioning them into perceptual, evaluative, cognitive and behavioural norms. These four types of norms are associated with four distinct attitudes, respectively (Stamper, 1996):

- Ontological ñ to acknowledge the existence of something;
- Axiologic ñ to be disposed in favour or against something in value terms;
- Epistemic ñ to adopt a degree of belief or disbelief;
- Deontic ñ to be disposed to act in some way.

An EDA agent is a knowledge-based system whose knowledge base structure, depicted in figure 2, is based on three components: the Epistemic, the Deontic and the Axiologic.

The epistemic model component is where the knowledge of the agent is stored, in the form of statements that are accepted by that agent. Two types of knowledge are stored here: declarative knowledge ñ statements relative to the agent beliefs ñ and procedural knowledge ñ statements concerning the know-how of the agent, *e.g.* their plans and procedural abilities.

The importance of norms to action has determined the name we have chosen for the model component where the agent goals are represented. An agent goal may be simply understood as the *desire to perform an action* (which would motivate the designation of *conative*) but it can also be understood, especially in a social context, as the *result of the internalisation of a duty or social obligation* (which would motivate the designation of *deontic*). We have adopted the latter designation not only because we want to emphasise the importance of social obligations but also because personal desires can be seen as a form of ëgeneralisedí obligation established by an agent for himself. This provides a *unification* of social and individual drives for action, which simplifies many aspects of the model.

The axiologic model component contains the value system of the agent, namely a partial order that defines the agent preferences with respect to norms[1]. The importance of the agent value system is apparent in situations of conflict, when it is necessary to violate a norm. This preference ordering is dynamic, in the sense that it may change whenever the other internal components of the agent model change, reflecting different beliefs or different goals.

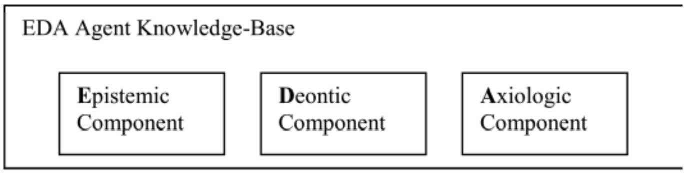


Fig. 2. The EDA agent knowledge-base structure

[1] It is important to note that these *values* are different from the values used in the evaluation process required to make a rational decision. The latter can been done in the scope of BDI models as illustrated in (Antunes and Coelho, 1999-1&2).

3 Intentions and Social Norms in the EDA Model

The multi-agent system metaphor that we have adopted for modelling organisations implies that organisations are seen as goal-governed collective agents, which are composed of individual agents. This perspective comes in line with the principles of normative agents proposed in (Castelfranchi et al., 1999).

In our model individual agents are autonomous, heterogeneous, rational, social agents. Therefore, they are compelled to make decisions and act in a way that, although not entirely deterministic, is constrained by their rationality. Actually their behaviour would be predictable if we knew all the details of their EDA model components, the environment stimuli, their perception function and also the reasoning machine they use, because the ultimate goal of a rational agent is to maximize its utility.

Typically, Artificial Intelligence (AI) agent models consider intelligent agent decision processes as internal processes that occur in the mind and involve exclusively logical reasoning, external inputs being essentially data that are perceived directly by the agent. This perspective does not acknowledge any social environment whatsoever. In this thesis we start from a totally different perspective, by emphasising the importance of social influences and a shared ontology on the agent decision processes, which then determines agentsí activity. We shall henceforth refer to agent as ëití although the EDA model also applies to human agents. In any case, we are particularly interested in the situations where information systems are formally described, thus making it possible for artificial agents to assist or replace human agents.

An important role of norms in agent decision processes is the so-called cognitive economy: by following norms the agent does not have to follow long reasoning chains to calculate utilities ñ it just needs to follow the norms.

However, instead of adopting a whole-hearted social sciences perspective, which is often concerned merely with a macro perspective and a statistical view of social activity, we have adopted an intermediate perspective, where social notions are introduced to complement the individualistic traditional AI decision models: a psycho-social perspective, whereby an agent is endowed with the capability of overriding social norms by intentionally deciding so.

Our model enables the relationship between socially shared beliefs with agent individual, private, beliefs; it also enables the analysis of the mutual relationships between moral values at the social level with ethical values at the individual level. However, we have found particularly interesting analogies in the deontic component, specifically in the nature of the entities and processes that are involved in agent goal-directed behaviour, by inspecting and comparing both the social processes and individual processes enacted in the deontic component of the EDA model.

This was motivated by the close relationship between deontic concepts and agency concepts, and represents a direction of research that studies agency in terms of normative social concepts: obligations, responsibilities, commitments and duties. These concepts, together with the concepts of power/influence, contribute to facilitate the creation of organisational models, and are compatible with a vision of organisations as normative information systems as well as with the notion of information field that underlies the organisational semiotics approach.

As will be explained in more detail in the next section, an essential aspect of the EDA model is that the Deontic component is based on the notion of *generalised goal* as a kind of obligation, that encompasses both social goals (social obligations) and individual goals (self obligations). This idea is inspired by Shohamís work (1993). Following a traditional designation in DAI, we designate those individual *generalised goals* that are inserted in the agenda as *achievement goals,* as in (Cohen and Levesque, 1990). Figure 3 describes the parallelism between mental and social constructs that lead to setting a goal in the agenda, and which justifies the adoption of the aforementioned generalised obligation. Here, p represents a proposition (world state). $B_{\alpha}(p)$ represents p as one of agent α ís beliefs. $O_{\alpha}^{\beta}(p)$ represents the obligation that α must see to it that p is true for β . $O_{\alpha}^{\alpha}(p)$ represents the interest that α has on seeing to it that p is true for itself ñ a kind of self-imposed obligation. In this diagram $p \in \mathrm{E}_{\alpha}(W,D)$ means, intuitively, that proposition p is one of the goals on α ís agenda.

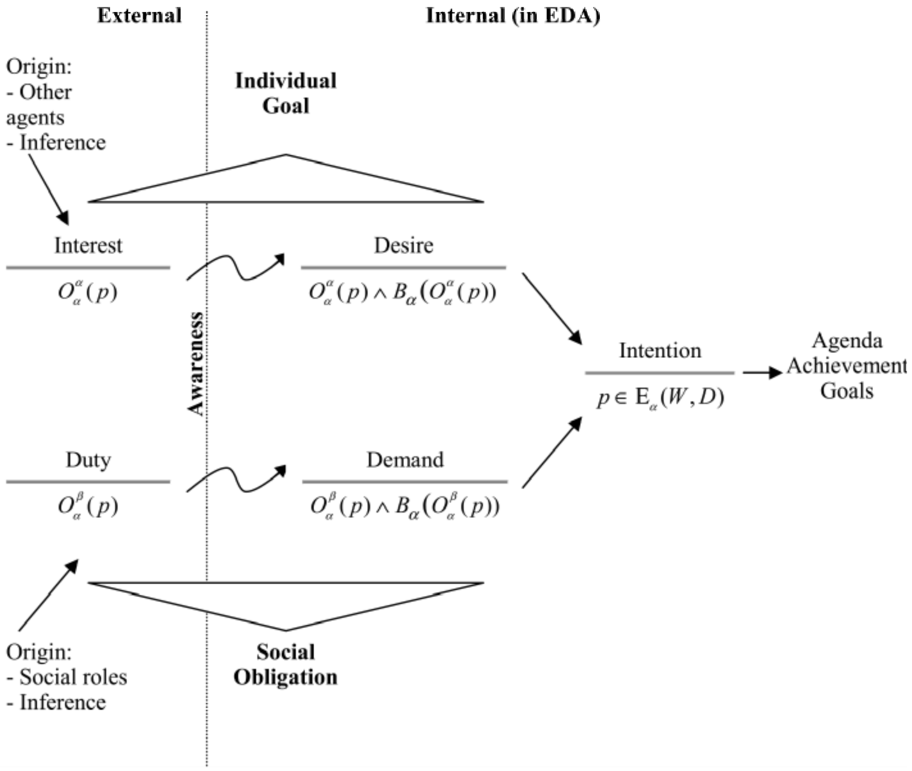


Fig. 3. Social and Individual goals parallelism in the EDA model

Interest is one of the key notions that are represented in the EDA model, based on the combination of the deontic operator ëought-to-beí (von Wright, 1951) and the agentive ësee-to-it-thatí *stit* operator (Belnap, 1991). *Interests* and *Desires* are manifestations of *Individual Goals*. The differences between them are the following:

- *Interests* are individual goals of which the agent is not necessarily aware, typically at a high abstraction level, which would contribute to improve its overall utility. Interests may be originated externally, by other agentsí suggestions, or internally, by inference: deductively (means-end analysis), inductively or abductively. One of the most difficult tasks for an agent is to become aware of its interest areas because there are too many potentially advantageous world states, making the full utility evaluation of each potential interest impossible, given the limited reasoning capacity of any agent.
- *Desires* are interests that the agent is aware of. However, they may not be achievable and may even conflict with other agent goals; the logical translation indicated in the figure, $O_{\alpha}^{\alpha}(p) \wedge B_{\alpha}(O_{\alpha}^{\alpha}(p))$, means that desires are goals that agent α ought to pursue for itself and that it is aware of. However, the agent has not yet decided to commit to it, in a global perspective, *i.e.* considering all other possibilities. In other words, desires become intentions only if they are part of the preferred extension of the normative agent EDA model (Filipe, 2000).

It is important to point out the strong connection between these deontic concepts and the axiologic component. All notions indicated in the figure should be interpreted from the agent perspective, *i.e.* values assigned to *interests* are determined by the agent. Eventually, external agents may consider some goal (*interest*) as having a positive value for the agent and yet the agent himself may decide otherwise. That is why *interests* are considered here to be the set of all goals to which the agent would assign a positive utility, but which it may not be aware of. In that case the responsibility for the *interest* remains on the external agent.

Not all interests become desires but all desires are agent interests. This may seem contradictory with a situation commonly seen in human societies of agents acting in *othersí best interests*, sometimes even against their desires: thatís what parents do for their children. However, this does not mean that the agent desires are not seen as positive by the agent; it only shows that the agent may have a deficient axiologic system (by its information field standards) and in that case the social group may give other agents the right to override that agent. In the case of artificial agents such a discrepancy would typically cause the agent to be banned from the information field (no access to social resources) and eventually repaired or discontinued by human supervisors, due to social pressure (*e.g.* software viruses).

In parallel with *Interests* and *Desires*, there are also social driving forces converging to influence individual achievement goals, but through a different path, based on the general notion of social obligation. Social obligations are the goals that the social group where the agent is situated require the agent to attain. These can also have different flavours in parallel to what we have described for individual goals.

- *Duties* are social goals that are attached to the particular roles that the agent is assigned to, whether the agent is aware that they exist or not. The statement

$O_\alpha^\beta(p)$ means that agent α ought to do p on behalf of another agent β. Agent β may be another individual agent or a collective agent, such as the society to which α belongs. Besides the obligations that are explicitly indicated in social roles, there are additional implicit obligations. These are inferred from conditional social norms and typically depend on circumstances. Additionally, all specific commitments that the agent may agree to enter also become duties; however, in this case, the agent is necessarily aware of them.

- *Demands* are duties that the agent is aware of[2]. This notion is formalised by the following logical statement: $O_\alpha^\beta(p) \wedge B_\alpha(O_\alpha^\beta(p))$. Social demands motivate the agent to act but they may not be achievable and may even conflict with other agent duties; being autonomous, the agent may also decide that, according to circumstances, it is better not to fulfil a social demand and rather accept the corresponding sanction. Demands become intentions only if they are part of the preferred extension of the normative agent EDA model ñ see (Filipe, 2000 section 5.7) for details.
- *Intentions*: Whatever their origin (individual or social) intentions constitute a non-conflicting set of goals that are believed to offer the highest possible value for the concerned agent. Intentions are designated by some authors (Singh, 1990) as psychological commitments (to act). However, intentions may eventually (despite the agent sincerity) not actually be placed in the agenda, for several reasons:
  - They may be too abstract to become directly executed, thus requiring further means-end analysis and planning.
  - They may need to wait for their appropriate time of execution.
  - They may be overridden by higher priority intentions.
  - Required resources may not be ready.

The semantics of the prescriptive notions described above may be partially captured using set relationships as depicted in figure 4, below.

When an agent decides to act in order to fulfil an intention, an agenda item is created ñ we adopt the designation of *achievement goal*. Achievement goals are defined as in (Cohen and Levesque, 1990) as goals that are shared by individuals participating in a team that has a *joint persistent goal*. Following the terminology of (Cohen and Levesque, 1990) agent α has a *weak achievement goal*, relative to its motivation (which in our case corresponds to the origin and perceived utility of that goal), to bring about the joint persistent goal γ if either of the following is true:

- α does not yet believe that γ is true and has γ being eventually true as a goal (*i.e.* α has a normal achievement goal to bring about γ)

[2] According to the Concise Oxford Dictionary, *demand* is ì*an insistent and peremptory request, made as of right*î. We believe this is the English word with the closest semantics to what we need.

- α believes that γ is true, will never be true or is irrelevant (utility below the motivation threshold), but has a goal that the status of γ be mutually believed by all team members.

However we do not adopt the notion of joint persistent goal for social co-ordination, as proposed by Cohen and Levesque (1990) because their approach has a number of shortcomings, not only theoretical but also related to the practical feasibility of their model, which are well documented in (Singh, 1996).

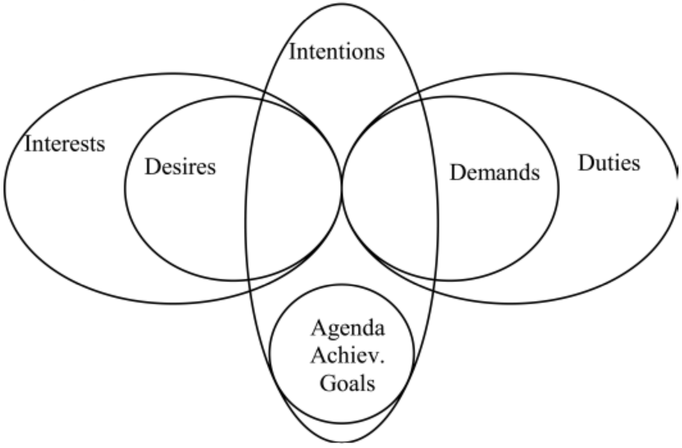


Fig. 4. Set-theoretic relationships among deontic prescriptive concepts

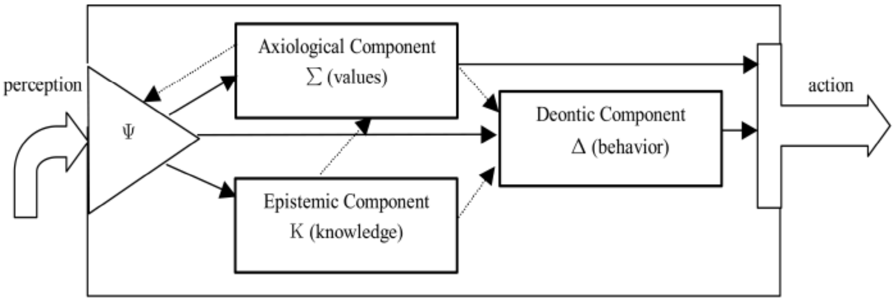


Fig. 5. The EDA model component relationships

4 The EDA Model Internal Architecture

Using the social psychology taxonomy of norms, and based on the assumption that organisational agentsí behaviour is determined by the evaluation of deontic norms, given the agent epistemic state, with axiological norms for solving eventual interest conflicts, we propose an intentional agent model, which is decomposed into three main components: the epistemic, the deontic and the axiologic. Additionally there are two external interfaces: an input (perceptual) interface, through which the agent receives and pragmatically interprets messages from the environment and an output (acting) external interface through which the agent acts upon the environment, namely sending messages to other agents[3].

A socially shared ontology is partially incorporated in an agent cognitive model whenever it is needed, *i.e.* when the agent needs to perform a particular role. In this case, beliefs are incorporated in the Epistemic component, obligations and responsibilities are incorporated in the Deontic component and values (using a partial order relation of importance) are incorporated in the Axiologic component ñ all indexed to the particular role that the agent is to play.

Figure 5 depicts the EDA model and its component relationships.

! is a pragmatic function that filters perceptions, according to the agent ontology, using perceptual and axiologic norms, and updates one or more model components.

∀ is an axiologic function that is used mainly in two circumstances: to help decide which signs to perceive, and to help decide which goals to put in the agenda and execute.

is a knowledge-based component, where the agent stores its beliefs both explicitly and implicitly, in the form of potential deductions based on logical reasoning.

∃ is a set of plans, either explicit or implicit, the agent is interested in and may choose to execute.

The detailed description of each component, including its internal structure, is provided in (Filipe, 2000). In this paper we focus on the system behaviour. The next sections show how in EDA we specify ideal patterns of behaviour and also how we represent and deal with non-ideal behaviours.

5 Organisational Modelling with Multi-agent Systems Using the EDA Model

The EDA model may apply to both human and artificial agents, and is concerned with the social nature of organisational agents:

- Firstly, because it accounts for a particular mental structure (Epistemic-Deontic-Axiologic) that is better, for our purposes, than other agent mental structures proposed in the literature to model agent interaction. Specifically, we intend to use it for modelling information fields where social norms influence individual agents and are used by an agent to guide inter-subjective communication and achieve multi-agent co-ordination.

[3] In this thesis we restrict our attention to the semiotic, symbolic, types of agent activity, ignoring substantive physical activities.

- Secondly, because the model is based on normative notions that are not only intended to guide the agent behaviour in a socially accepted way, but also to identify what sanctions to expect from norm violations, in order to let the agent take decisions about its goals and actions, especially when co-ordination is involved. The EDA model is based on the claim that multi-agent notions such as social commitment, joint intentions, teamwork, negotiation and social roles, would be merely metaphorical if their normative character were not accounted for.

Given its social-focused nature, the EDA agent notion may be used to model and implement social activities, involving multi-agent co-ordination. However, although the agent paradigm described in this thesis is suited to model team work and joint problem solving, the major novelty with respect to other current agent models is the normative flavour. EDA agents are able to co-ordinate on the basis of shared norms and social commitments. Shared norms are used both for planning and for reasoning about what is expected from other agents but, internally, EDA agents keep an intentional representation of their own goals, beliefs and values.

Co-ordination is based on commitments to execute requested services. Commitments are represented not as joint intentions based on mutual beliefs, as is the case of the Cohen-Levesque model, upon which the BDI paradigm is based, but as first-class social concepts, at an inter-subjective level. The organisational memory embedded in the representation of socially accepted best-practices or patterns of behaviour and the handling of sub-ideal situations is, in our opinion, one of the main contributions that a multi-agent system can bring about.

6 Representing Ideal and Non-ideal Patterns of Behaviour

The EDA model is a norm-based model. Norms are ultimately an external source of action control. This assumption is reflected specially in the Deontic component of the EDA model.

Standard Deontic Logic (SDL) represents and reasons about ideal situations only. However, although agent behaviour is guided by deontic guidelines, in reality an agent who always behaves in an ideal way is seldom seen. The need to overcome the limited expressiveness of SDL, and to provide a way to represent and manipulate sub-ideal states has been acknowledged and important work has been done in that direction, *e.g.* by Dignum *et al.* (1994).

Contrary to SDL, the Deontic component of the EDA model is designed to handle sub-ideal situations. Even in non-ideal worlds, where conflicting interests and obligations co-exist, we wish to be able to reason about agent interests and desirable world states in such a way that the agent still is able to function coherently.

Behaviours may be represented as partial plans at different abstract levels. A goal is a very high abstract plan, whereas a sequence of elementary actions defines a plan at the instance level. The Deontic component is similar, in this sense, to what Werner (1989) called the agent intentional state.

However, in our model, agent decisions depend both on the available plans and a preference relationship defined in the axiologic component. This value assignment, which is essential for determining agent intentions, *i.e.* its preferred actions, can change dynamically, either due to external events (perception) or to internal events (inference), thus dynamically modifying the agentís intentions.

Although our representation of ideal behaviours is based in deontic logic, we acknowledge the existence of problems with deontic logic, partially caused by the fact that the modal ëoughtí operator actually collapses two operators with different meanings, namely ëought-to-doí and ëought-to-beí. Our solution, inspired in (Hilpinen, 1971) and (Belnap, 1991), is to use a combination of action logic and deontic logic for representing agentive ëought-to-doí statements, leaving the standard deontic operator for propositional, declarative, statements. Agentive statements are represented as $[\alpha\ stit : Q]$ where α stands for an agent and *Q* stands for any kind of sentence (declarative or agentive). An ë*ought-to-do*í is represented using the conventional ë*ought-to-be*í modal operator combined with an agentive statement, yielding statements of the form $O[\alpha\ stit : Q]$ or, for short: $O_\alpha(Q)$.

The representation of behaviours using this kind of agentive statement has several attractive properties, including the fact that it is a declarative representation (with all the flexibility that it provides) and that there exists the possibility of nesting plans as nested *stits*.

A plan is typically given to the agent at a very abstract level, by specifying the goal that it ought to achieve. The agent should then be able to decompose it into simple, executable, actions. This decomposition can be achieved by a means-ends process. By representing plans declaratively, as behavioural norms, the process becomes similar to backward chaining reasoning from abstract goals to more specific ones, until executable tasks are identified, the same way as in goal oriented reasoning in knowledge-based systems. This similarity enables the adoption of methods and tools from that area, being especially useful the inference engine concept.

However, the Deontic component does not control the Agenda directly, *i.e.* it is not responsible for setting the agent goals directly, because the prospective agent goals ñ similar to *desires*, in the BDI model ñ must be analysed by the Axiologic component first, which computes their value accordingly to an internal preference relation, taking into account possible obligation violations.

7 Generalised Goals

Goal-governed behaviour can be represented, according to agency logic (Belnap, 1991), by an agentive statement where an agent α sees to it that the state of affairs *Q* obtains at time $\%_2$ by the same agent α performing a choice at time $\%_1$: $[\alpha\ stit : Q]$. We claim that it is possible to use the same syntax and semantics both for representing goals derived from both agent interests and behavioural social norms. This would make it possible to build up detailed plans from agent interests and behavioural norms using means-ends analysis ñ the same inference mechanism commonly used in knowledge-based systems for goal oriented backward-chaining.

We propose an unification of different concepts under the notion of generalised goal. This unification stems from the following axioms:

- All actions have a responsible agent, who performs the action.
- The performer may not be the ultimate responsible for the action and, in the case of artificial agents, it never is: the last agent in the responsibility chain must be a person.
- An action is performed for satisfying someoneís interest.
- If the client for whom the action is performed is not satisfied he can ask for a sanction to be imposed by the information field that defines the context of the action.

We propose the following formal representation for generalised goals:

$$\mathrm{G_i} = O^{\beta/\chi}_{\alpha/\rho}(Q,\tau,\sigma) \supset O\,(\left[\alpha\ \mathit{stit}: Q\right] \text{in-time-window } \tau \text{ subject-to-sanction } \sigma\,)$$

This can be read as: α (under the responsibility of ρ) has the goal of ensuring Q, in reply to a request of β (under the control of χ); α is the performing agent, β is the client that requested the service, χ is a controller agent ñ which Castelfranchi (1993) calls a witness, Singh (1996) calls a context, and we associate to the information field. ρ is the responsibility chain for α[4]; O is the standard deontic operator ëought-to-beí; $\left[\alpha\ \mathit{stit}: Q\right]$ is an agency statement, saying that agent α sees-to-it-that proposition Q becomes true. This means that α will perform a plan to bring about Q in time window %, where %is a time expression specifying the time window during which proposition Q is intended to be satisfied; %may be specified in absolute time or relative to some event; &indicates the sanction cost of violation.

If agent α is different from agent β then we have a social commitment; otherwise we have an intention. In the latter case there is no controller agent χ and the sanction σ is typically null (represented by $\varnothing$) with expected value zero.

8 Conclusions

The EDA model is a norm-based, theoretically well-founded in social psychology, agent model that takes into account not only the intentional aspects of agency but also the social norms that prescribe and proscribe certain agent patterns of behaviour. The main components of this model (Epistemic, Deontic and Axiological) have a direct relationship with the types of norms that are proposed in the social psychology theory supporting the model. In this paper, however, we focused our attention essentially in the Deontic component, where the normative social aspects are more important, namely where ideal and sub-ideal behaviours are represented.

[4] The responsibility chain is a list constituted by one or more agents that are responsible for the commitment to service performance and may be brought to scene if there is a breakdown during the service performance or if the performing agent is unable to handle the service satisfactorily.

We consider agents to be goal-governed systems. Using the *generalised goal* concept, all agent goals can be represented as obligations, encompassing both agent self-imposed obligations and social obligations ñ derived from moral obligations or commitments established in the course of their social activity. The proposed modal operator ñ G_i ñ overcomes the excessive strength of *stit*, and implicitly permits the representation of an important organisational activity: delegation, which Belnapís *stit* does not allow. Furthermore, the inclusion of a ë*committed-toí* agent also extends *stit* and permits the bringing in of the notion of social commitment, based on the notion of obligation towards other agents and subject to sanctions.

Instead of manipulating the notions of belief, intention, desire, commitment, goal and obligation, the only primitives that we need are *belief* and *generalised goal*. All other notions are constructed from these ones. The interested reader can find more details about this in (Filipe, 2000).

References

[1] Antunes, L. and H. Coelho, 1999. Rationalising decisions using multiple values, *Proceedings of the European Conference on Cognitive Science*, Siena, Italy.

[2] Antunes, L. and H. Coelho, 1999. Decisions based upon multiple values: the BVG agent architecture. *Proceedings of the Portuguese AI Conference EPIA99*, Springer-Verlag LNAI (Lecture Notes in Artificial Intelligence) n∫ 1695, pp. 297-311, ..vora, Portugal.

[3] Belnap, N., 1991. Backwards and Forwards in the Modal Logic of Agency. Philosophy and Phenomenological Research, vol. 51.

[4] Castelfranchi, C., 1993. Commitments: from Individual Intentions to Groups and Organisations. In Working Notes of AAAIí93 *Workshop on AI and Theory of Groups and Organisations: Conceptual and Empirical Research*, pp.35-41.

[5] C. Castelfranchi, F. Dignum, C. Jonker and J. Treur, 1999. Deliberate Normative Agents: Principles and Architectures. In N. Jennings and Y. Lesperance (eds.) Proceedings of ATAL-99, Orlando, USA.

[6] Cohen, P. and H. Levesque, 1990. Intention is Choice with Commitment. *Artificial Intelligence*, **42**:213-261.

[7] Conte, R., and C. Castelfranchi, 1995. *Cognitive and Social Action*, UCL Press, London.

[8] Dignum, F., J-J Ch. Meyer and R.Wieringa, 1994. A Dynamic Logic for Reasoning About Sub-Ideal States. In J.Breuker (Ed.), *ECAI Workshop on Artificial Normative Reasoning,* pp.79-92, Amsterdam, The Netherlands.

[9] F. Dignum, D. Morley, E. Sonenberg, and L. Cavendon. Towards socially sophisticated BDI agents. In Proceedings of the Fourth International Conference on MultiAgent Systems, ICMAS-2000, Boston, USA.

[10] Filipe, J., 2000. Normative Organisational Modelling using Intelligent Multi-Agent Systems. Ph.D. thesis, University of Staffordshire, UK.

[11] Hilpinen, R., 1971. *Deontic Logic: Introductory and Systematic Readings*. Reidel Publishing Co., Dordrecht, The Netherlands.

[12] Rao, A. and M. Georgeff, 1991. Modeling Rational Agents within a BDI architecture. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning* (Nebel, Rich, and Swartout, Eds.) pp.439-449. Morgan Kaufman, San Mateo, USA.

[13] Shoham, Y., 1993. Agent-Oriented Programming. *Artificial Intelligence,* **60**, pp.51-92, Elsevier Science Publishers.

[14] Stamper, R., 1996. Signs, Information, Norms and Systems. In Holmqvist et al. (Eds.), Signs of Work, Semiosis and Information Processing in Organizations, Walter de Gruyter, Berlin, New York.

[15] Singh, M., 1990. *Towards a Theory of Situated Know-How.* 9th European Conference on Artificial Intelligence.

[16] Singh, M., 1996. Multiagent Systems as Spheres of Commitment. Proceedings of the International Conference on Multiagent Systems (ICMAS) - Workshop on Norms, Obligations and Conventions. Kyoto, Japan.

[17] Werner, E., 1989. Cooperating Agents: A Unified Theory of Communication and Social Structure. In Gasser and Huhns (Eds.), *Distributed Artificial Intelligence*, pp.3-36, Morgan Kaufman, San Mateo, USA.

[18] von Wright, G., 1951. Deontic Logic. Mind, 60, pp.1-15.

Simulating Computational Societies

Lloyd Kamara, Alexander Artikis, Brendan Neville, and Jeremy Pitt

Intelligent & Interactive Systems Group, Imperial College
Electrical Engineering Dept., London, UK, SW7 2BT
Tel.: +44 (0)20 7594 6221
{l.kamara,a.artikis,brendan.neville,j.pitt}@ic.ac.uk

Abstract. Multi-agent systems can be considered from a variety of perspectives. One such perspective arises from considering the architecture of an agent itself. Another is that of an instantiated agent architecture and its interaction with its peers in a MAS. A third perspective is that of an external observer. These three perspectives cover a potentially overlapping but essentially distinct set of issues concerning MAS simulation and modelling. In this paper, we consider each of these perspectives in turn and demonstrate how a simulation framework can support a collective treatment of such concepts. We discuss the implications for agent development and agent society design arising from the results and analysis of our simulation approach.

1 Introduction

Multi-agent systems (MAS) can be considered from a variety of perspectives. One such perspective arises from considering the architecture of an agent itself (e.g. [1]). This is relevant to agent designers and implementors, as it concerns the *internal operation* of an agent. The basic architectural properties largely determine how the agent will interact with its environment. Another perspective to consider is that of an instantiated agent architecture and its interaction with its peers in a MAS. This covers both *communicative* and *socio-cognitive* aspects of agent interaction (e.g. [2, 3, 4]). In addition to being relevant to agent designers and implementors, this can be used to model, analyse and explain the behaviour of the agents in terms of *social theories* (e.g. [3, 4]). The introduction of *socio-cognitive elements* to agent reasoning allows interactions to be characterised in increasingly anthropomorphic terms. A third perspective is that of an external observer (e.g. [5, 6, 7, 8]). Under such circumstances, we adopt a bird's eye view of computational systems — we are not concerned with the internal architecture of the participating agents. We can specify the legal and social aspects of these systems, such as the *normative positions* and *institutional powers* of the agents, without making any assumptions about mentalistic concepts like beliefs, desires or intentions.

We aim to enable society designers and agent developers to view simulated societies both at a micro (i.e. from an intensional perspective) and a macro

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 53–67, 2003.

(i.e. from an extensional perspective) level (as is the case with [9] and [7])[1]. To this end, we propose simulation tools which address concerns and requirements identified at both (and intermediate) levels. We refer to these tools collectively as our *simulation framework*.

Pitt *et al.* [11] describe an abstract producer/consumer (APC) scenario where producers sell information to consumers. In this scenario the producers are explorer agents that map out the distribution of oil in their environment and consumers are cartographer agents that initiate contract-net protocols [12] (CNP) to acquire the maps from the explorers. We have used variants of the CNP in Section 3 and Section 4 as vehicles for analysing trust, experience and reputation from the socio-cognitive (micro) perspective; and power, permission and obligation from the external (macro) perspective.

The rest of this paper is organised as follows. In Section 2, we present MAS and agent simulation architectures. In Section 3, we describe an approach to trust and reputation modelling in MAS. In Section 4, we consider executable specifications of norm-governed computational societies in a manner independent of agent internals. In Section 5, we relate our findings to agent development and agent society design endeavours, concluding that they offer some preliminary guidelines to agent implementors and society developers for scoping social MAS.

2 Agent and MAS Simulation Architectures

The basis of our MAS simulation architecture is a collection of agents whose interaction is governed by a *communications interface* and an accompanying set of *protocols*. The MAS is neutral with respect to the architecture of participating agents: no assumptions are made about internal operation and motivations of an agent by its peers. Interaction between two agents takes place through their communications interfaces, either creating or extending a communicative context, or *conversation*.

2.1 Communications Interface

The communications interface consists of a higher and lower-level component pair: the Agent Communication Language (ACL) and a socket-based module for TCP/IP transmission of messages. The former defines (amongst other things) the supported syntax of messages while the latter defines how messages are exchanged. Following [2], we define an ACL in terms of three components: a *content language*, which defines the syntax of messages, a set of *protocols* which define patterns of interaction and a *reply function* which provides an external semantics for the ACL. The content language is the set of performatives that agents use in communication, while the protocols identify sequences of performatives

[1] We also note that these intensional and extensional perspectives are respectively related to the subjective and objective approaches to multi-agent co-ordination dicussed in [10].

that constitute meta-level interaction (such as auction and query/response protocols). Given an input performative, protocol and current conversation state, the reply function returns the set of 'acceptable' responses. These take the form of performative-protocol pairs, which, depending on the options available, may enable a responding agent to continue the conversation using the same protocol, or to initiate an auxiliary conversation using a different protocol. The relationship between content language, protocols and reply function is specified in a separate (`Prolog`) module, making it straightforward to refine and enhance the communicative behaviour of agents.

Pitt and Mamdani [2] justify the use of a protocol-based semantics to cover the external aspects of agent interaction in MAS featuring agents with heterogeneous architectures and distinct reasoning and motivational traits. A similar justification is used here, with the introduction of a socio-cognitive perspective arguably serving to re-inforce the need for such an approach. The additional subjective treatment of concepts including trust and reputation makes each agent even more distinct and thus external characterisations of interactions become increasingly relevant. In the following section, we describe an agent architecture that can be parameterised with different reasoning and socio-cognitive behaviour, allowing us to represent heterogeneous MAS in our experiments while retaining a compact simulation base.

2.2 Agent Architecture

In our proposed experimental configuration, a generic BDI-based [1] architecture forms the computational base for each agent. We use the same architecture for reasons of simplicity: the principle of heterogeneity outlined earlier has not been abandoned. It would be equally possible to use agents with different architectures as long as they possess compatible interfaces (like in [13, 14]). Specific behaviour is determined by parameterisation of the respective agent instances. This approach can be used, for example, to pre-assign the roles of auctioneer and bidders in a group of agents enacting an electronic auction. Figure 1 gives a diagrammatic overview of the agent architecture. The control module contains the agent interface and interpreter, in addition to housing the protocols sub-module. The agent interface facilitates and manages conversations (as mentioned previously), representations of which are to be found in the conversational component of the agent's mental state.

The interpreter serves as the animating core of the agent architecture, operating in the form of a typical BDI cycle. It processes incoming messages at the content level, consulting and updating mental state representation while executing communicative actions appropriate to the agent's recent perception and interpretation of events.

The mental state representation consists of several elements (see Figure 1). The intentional state holds the agent's belief and motivational state alongside the agent's current intentions. These are expressed as time-stamped `Prolog` terms with assertion being the means of updating state content. Records of conversations — again, represented through `Prolog` terms — are maintained in the

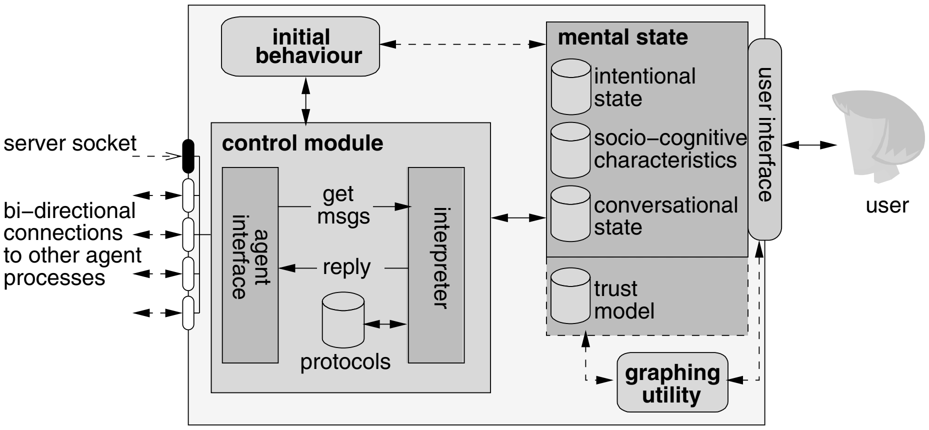


Fig. 1. Generic agent architecture for experiments

conversational state component. Each conversation is recorded as a four-tuple denoting the correspondent (the other agent in the conversation), local and remote conversation identifiers [2] and the current state of the conversation (from a local perspective). A sub-set of the contained data reflects proceedings in current, or active, conversations. The user interface provides a means for the experimenter to initialise the agent and to monitor and control its behaviour. The architecture does not feature a fixed procedure for belief update at present, although the way has been prepared for the introduction of such a mechanism. In particular, with each `Prolog` term used to represent a belief there is an associated credence measure (in the range of 0–1) which reflects the level of certainty the agent has in the content.

3 Socio-cognitive Modelling

Our investigation of socially motivated behaviour is based on formal models of anthropomorphic socio-cognitive relations. Accordingly, our socio-cognitive model defines three component beliefs of each agent about its peers in the society: *trust*, *direct experience* and *reputation*. This section defines the socio-cognitive model and the software developed to facilitate experimentation with agent societies whose members are implemented with a (parameterised) version of the model.

Our computational representation of trust is based on the formal model of Castelfranchi and Falcone [3, 4]. The essential conceptualisation is as follows: the degree to which Agent A trusts Agent B about task τ in Ω (state of the world) is a subjective probability; this is the basis of agent A's decision to rely upon B for τ. Our method incoporates this stance, defining trust as the resultant belief of one agent about another, borne out of direct experience of that other party and/or from the testimonies of peers (i.e. reputation).

We define direct experience as the belief one agent has about the trustworthiness of another, based on first-hand interactions. Having 'trusted' another agent to perform task τ and assessed the outcome, an agent will update its direct experience beliefs concerning the delegated agent accordingly. The outcome of delegating a task may be either successful or unsuccessful: this categorisation is the basis of an experience update rule [15] that calculates the revised level of trustworthiness to associate with the agent in question.

Our motivation in investigating reputation mechanisms by simulation is provided by Conte [16], who outlines the need for 'decentralised mechanisms of enforcement of social order' noting that 'reputation plays a crucial role in distributed social control'. In an agent society, reputation is the collectively informed opinion held by a group of agents about the performance of a peer agent within a specific social context. By consulting its peers, an agent can discover the individual reputation of an agent. However, the received testimonies may be affected by existing relationships and attitudes (for example, agent C may be willing to divulge reputation information to agent A but not to agent B). Thus, reputation is also a subjective concept which we define as a belief held/derived by one agent. The Subjective Reputation Evaluation Function (**SREF**) formulates subjectively — that is, from the perspective of an agent A — the reputation of an agent B, based on information from the peers of A and B. **SREF** may take several forms; examples are a weighted sum or a fuzzy relation. In our current work, we use the former approach; a summing function in which n assertions received from peers regarding agent B are weighted by the agent's trust (confidence) in each peer to make accurate recommendations. This approach incorporates the credibility of each assertion's source (the credibility of a belief being dependent upon the credibility of its sources, evidences and supports [3]).

We briefly address a proposed method of combining direct experience with reputation to formulate the mental state of trust. Each belief has associated with it a degree of confidence signifying an agent's trust in the belief's accuracy. This confidence measure is essential to what trust will be based on, be it one or indeed both of the beliefs, experience or reputation. An agent with strong confidence in its experience beliefs and little confidence in the accuracy of its reputation beliefs should rationally choose to calculate its degree of trust (**DoT**) primarily from its experiences. An agent recently introduced to the system should have little confidence in its own (in)experience and should thus base its trust solely on reputation. The number of direct experiences is therefore significant in evaluating confidence in a belief about direct experience. It is similarly the case for reputation, where the number of agreeing testimonies is a decisive factor. Experience and reputation thus become influences of trust, the weighting assigned to them dependent upon an agent's confidence in their respective accuracies.

3.1 The Trading Economy

We simulate our commerce society as trading agents executing a variant of the CNP [12]. In this variant, the corresponding communicative acts for informing of the successful completion of a task and making payment are sent without any intermediate evaluation — that is, the cartographer (manager) does not know how well (if at all) a task has been performed before it sends a payment in response to a contracted explorer's (bidder) *task completed* message. In effect, we have a CNP concluding with a *prisoner's dilemma* [17] (PD). Each agent's actions at any stage will be influenced by expectations about the actions of its trading counterpart. The agents are able to execute the CNP repeatedly, forming the basis for an iterated PD. This differs from the game theoretic studies in [17], however, in that agents choose the peers with which they are willing to enter the PD on a trust/cost trade-off basis. The reasoning behind which action to take once a contract is established is similarly distinguished.

An explorer agent is able to vary bidding price from one CNP instance to another. This could be used to increase the desirability of an agent's bid, by under-cutting the bids of other explorers held in higher esteem. Agents not being awarded contracts may lower their prices in an attempt to generate business. Once a loyal trade relationship has been established, an explorer agent can gradually increase its bidding price in order to increase profitability.

When a cartographer awards a contract to a specific explorer, a PD-style dependence is obtained. The explorer is relying upon the cartographer to make prompt payment; the cartographer is relying upon the explorer to provide the required information. Our system however, does not guarantee that agent actions will always have the intended effect. Agents may fail to complete a task for reasons other than conscious deception. The inclusion of this *fallibility factor* is to account for unpredictability and unreliability within MAS environments and real-world applications.

3.2 Trust-Based Prediction

Here, we discuss the use of a trust mechanism as a predictor for future peer behaviour. In this context, we interpret trust as a subjective probability that a peer will take a certain action, the outcome of which will characterise the corresponding interaction context. For each outcome, the agent will experience profit or loss to a varying extent — we refer to this as the outcome's *pay-off*. Knowing the pay-offs of each outcome in advance and having an estimate of their probabilities of occurrence (trust), the agent and calculate the expected pay-off of each of its options. How the agent determines trading partners and what action to take (*co-operate* or *defect*) once a contract is established depends upon its character type and the expected pay-offs of each possible action. We currently simulate three basic character types. *Co-operators* only choose co-operation strategies; if the expected outcome for co-operation is not greater than zero, a co-operator will not enter into a contract. *Defectors* will pursue a policy of co-operation or defection purely based upon maximising the expected pay-off of an interaction. *Reciprocators* have probabilistic intentions; the probability that they will co-operate (their *trustworthiness*) is matched to their evaluation of a trading partner's behaviour (**DoT** in the peer). As a result, their trustworthiness displayed towards competent co-operators is high, whereas they are likely to defect against defectors and incompetent agents in general.

3.3 Agent Configuration and Monitoring

The *Multiple Agent Launcher* (MAL) is a software tool that allows experimenters to create and launch an arbitrary number of agents with distinct configuration parameters. Agents may be launched individually or in groups. The configuration parameters are expressed as arguments to the agent architecture presented in Section 2 and range from low-level (e.g. agent cycle delay time) to high-level (e.g. file-name of `Prolog` source for run-time behaviour). The motivation behind the MAL is to provide the user with a single interface from which to configure and monitor agent experiments, as well as to visualise real-time agent interactions. It is important to note that the agents thus launched remain semi-autonomous processes; the MAL is not a centralised control mechanism, but behaves as a convenient abstraction of one. The MAL supports configuration of agents with heterogeneous socio-cognitive characteristics and behavioural traits, by allowing the experimenter to adjust the parameters of the trust and reputation update functions (for example). The output of these functions is logged at various intervals during a simulation run and can also be graphed in real-time.

3.4 Simulation and Results

We now discuss an example simulation for trust update based on direct experience. The experiment consists of twelve agents, split evenly between managers and bidders. All of the agents are of reciprocator interaction type, differing only in their ability to complete the tasks relied upon by their respective contract partners. This ability is manifested through a probability parameter which determines the rate of successful task completion per agent. In this experiment, three ability levels were specified: 95%, 75% and 50%. Two cartographers and two explorers were all assigned the same probability parameter for each of the three levels, resulting in a tripartite ability configuration. When an agent fails to successfully perform its part of the contract, its partner perceives this and consequently reacts as if it were a conscious act of defection. In the graphs of Figure 2 and Figure 3, each line reflects the mean data values belonging to a pair of equally skilled agents of the same type. The mean data readings for two explorer agents whose degree of ability is 95%, for example, is labelled 'Explorers (95%)'. Time-points occur at ten second intervals; the current trust beliefs and monetary worth of the agents are logged at each point.
Figure 2 plots the level of trust directed towards different members, over the time of the simulation. The featured **DoT** is simply the mean of the opinions of all the members in the society who know the agent in question. Agents are configured to initially blindly trust their peers (**DoT** = 1.0). As the simulation progresses their opinions change dramatically — in particular, there is a drop in communal trust for the capable agents. This is because the incompetent agents do not trust the competent agents, while the capable agents reciprocate for the incompetents' failures by defecting against them. The competent agents have a very strong trust relationship among themselves but a very poor one with others — the average is

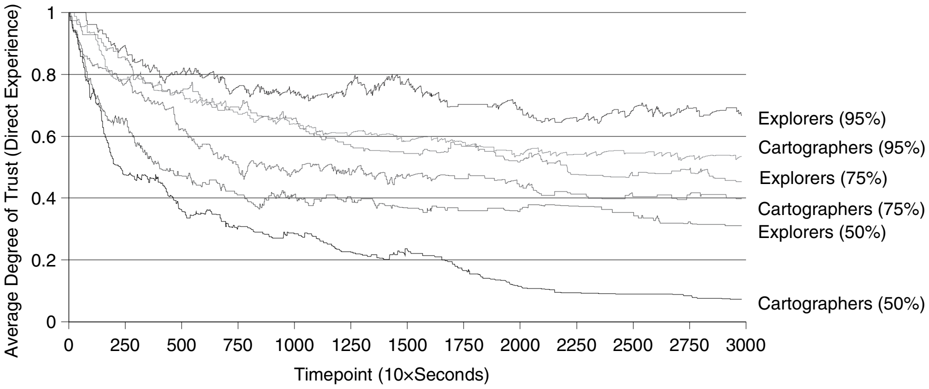


Fig. 2. Experimental results - Trust evolution

therefore lowered. Our next graph (Figure 3) shows the performance measure, the amount of assets accrued or lost. From this it can be seen that a society of reciprocators is a meritorious one. The highly capable agents (95%) are able to succeed; agents of intermediate skill (75%) break roughly even while the least skilled agents (50%) are punished.

Although our presented experiments have not covered reputation, we intend to address this in future work, as we believe it is an important aspect of agent interaction. In particular, we note how reputation is related to the notion of scalability: in a system with a large number of agents, the likelihood of any individual agent having interacted with a specific peer is less. This leads to an overall reduction in the amount of direct experience available to an agent during its decision-making. We are investigating when and how agents (in their capacity as third parties for reputation information transmission) can effectively

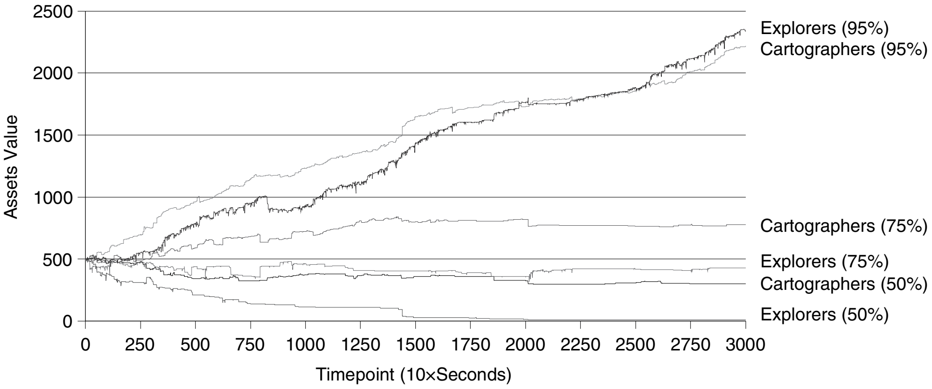


Fig. 3. Experimental results - Monetary performance

propagate information [16] under such circumstances. It is expected that the results of further experiments will establish what parameters and socio-cognitive characteristics lead to 'stable' trust relationships; ones that are resistant to minor aberrations in agent behaviour due to circumstances beyond their control.

4 Executing the Specifications of Computational Societies

Artikis *et al.* [5] present a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called *open computational societies*. Open computational societies (as defined in [5]) have the following characteristics: first, the internal architecture of the agents is not publicly known, i.e. there is no direct access to an agent's mental state. Second, the members of the society do not necessarily share a notion of global utility. Members may fail to, or choose not to, conform to specifications in order to achieve their individual goals. In addition to these properties, in open societies 'the behaviour of the members and their interactions cannot be predicted in advance' [18]. In this framework the computational systems are viewed from an external perspective, that is to say, the internal architecture of the agents is not considered. Three key components of computational systems are specified, namely the *social constraints*, *social roles* and *social states*. The specification of these concepts is based on and motivated by the formal study of legal and social systems (a theory of institutionalised power [19] and a theory of normative positions [20]) and traditional distributed computing techniques (state transition systems). The social constraints and roles have been specified with two different formalisms from AI: the Event Calculus [21] (see [5]) and the $C+$ language [22] (see [6]). Next, we briefly discuss the way social constraints are specified in [5, 6]. Then we present a computational framework that executes the specifications (i.e. the social constraints) of the agent societies during their execution.

4.1 Social Constraints

Artikis *et al.* [5, 6] follow Jones and Sergot [19] and maintain the standard, in the study of social and legal systems, long established distinction between *permission*, *physical capability* and *institutionalised power*. The social constraints specify:

- What kind of actions 'count as' [19] *valid* ('effective') actions. Distinguishing between *valid* and *invalid actions* enables the separation of meaningful from meaningless activities.
- What kind of actions (valid, invalid) are *permitted*. Determining the *permitted*, *prohibited*, *obligatory actions* enables the classification of the agent behaviour as 'legal' or 'illegal', 'social' or 'anti-social', etc.
- What are the *sanctions* and *enforcement policies* that deal with 'illegal', 'anti-social' behaviour.

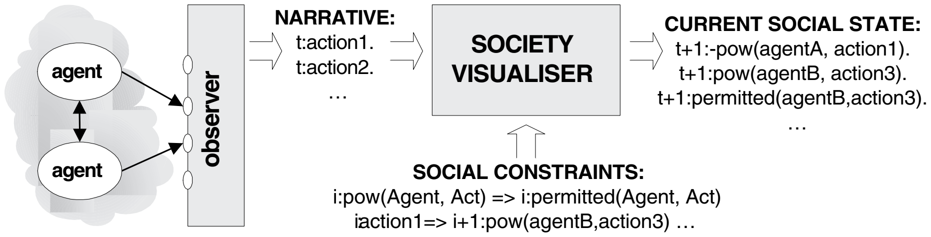

agent
agent
observer
NARRATIVE:
t:action1.
t:action2.
…
SOCIETY
VISUALISER
CURRENT SOCIAL STATE:
t+1:-pow(agentA, action1).
t+1:pow(agentB, action3).
t+1:permitted(agentB,action3).
…
SOCIAL CONSTRAINTS:
i:pow(Agent, Act) => i:permitted(Agent, Act)
i:action1=> i+1:pow(agentB,action3) …

Fig. 4. The Society Visualiser

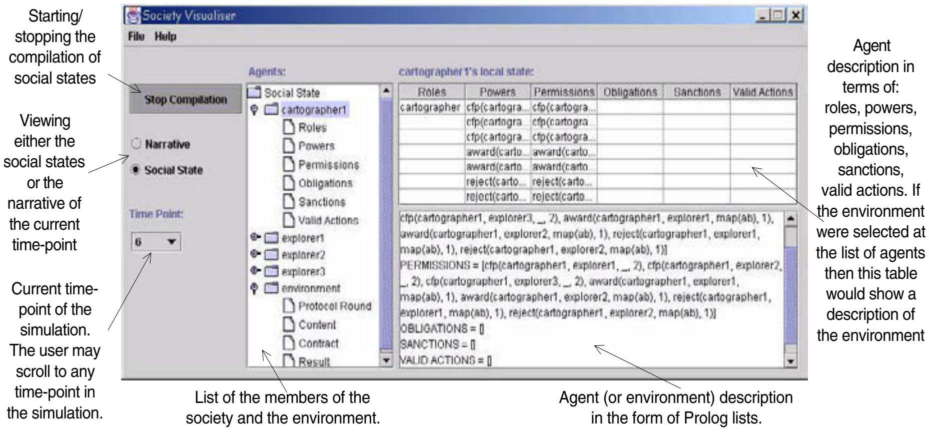
Valid actions are specified as follows: An action 'counts as' a *valid action* at a point in time if and only if the agent that performed that action had the *institutionalised power* [19] to perform it at that point in time. Differentiating between valid ('meaningful') and invalid ('meaningless') actions is of great importance in the analysis of agent systems. For example, in an auction, the auctioneer has to determine which bids are valid and therefore, which bids are eligible for winning the auction.

4.2 The Society Visualiser

In this section we describe a software platform called *Society Visualiser* (SV), that builds upon the theoretical framework for the specification of open systems [5] and that compiles (produces) and updates the global state of an agent society. The global state includes information like the institutional powers, permissions, obligations, sanctions and social roles of the members. Figure 4 describes the architecture of the SV in terms of inputs/outputs. In order to produce the global state of the societies, the SV has two inputs: a *narrative* and the *social constraints*[2]. The narrative is a description of the externally observable events that take place in a computational society. For example, t : *action1* states that *action1* took place at time point t (Figure 4). The narrative is produced in the following manner: When a member of the society sends a message to a peer, he also sends that message to an *observer agent* for monitoring purposes[3]. These messages are called *report* messages and are necessary for the compilation of the social states. The report messages (and all other monitored events) consist of the *narrative*. In other words, the SV observes (without intervening) the interactions of the members of the computational societies in order to produce the social states of the societies. The social constraints specify the valid actions, institutional powers, permissions and so on. of the members of the society. Consider

[2] The social constraints and the narrative are expressed in terms of either the Event Calculus or the $C+$ language. However, in order to simplify our analysis, we illustrate the social constraints and narrative in Figure 4 with the use of a generic notation.

[3] The observer agents monitors the execution of the members for the benefit of the Society Visualiser.



cartographer1's local state:

Roles	Powers	Permissions	Obligations	Sanctions	Valid Actions
cartographer	cfp(cartogra...	cfp(cartogra...			
	cfp(cartogra	cfp(cartogra			
	cfp(cartogra...	cfp(cartogra...			
	award(carto...	award(carto...			
	award(carto	award(carto			
	reject(carto...	reject(carto...			
	reject(carto...	reject(carto...			

```
cfp(cartographer1, explorer3, _, 2), award(cartographer1, explorer1, map(ab), 1),
award(cartographer1, explorer2, map(ab), 1), reject(cartographer1, explorer1,
map(ab), 1), reject(cartographer1, explorer2, map(ab), 1)]
PERMISSIONS = [cfp(cartographer1, explorer1, _, 2), cfp(cartographer1, explorer2,
_, 2), cfp(cartographer1, explorer3, _, 2), award(cartographer1, explorer1,
map(ab), 1), award(cartographer1, explorer2, map(ab), 1), reject(cartographer1,
explorer1, map(ab), 1), reject(cartographer1, explorer2, map(ab), 1)]
OBLIGATIONS = []
SANCTIONS = []
VALID ACTIONS = []
```

Starting/ stopping the compilation of social states

Viewing either the social states or the narrative of the current time-point

Current time-point of the simulation. The user may scroll to any time-point in the simulation.

List of the members of the society and the environment.

Agent (or environment) description in the form of Prolog lists.

Agent description in terms of: roles, powers, permissions, obligations, sanctions, valid actions. If the environment were selected at the list of agents then this table would show a description of the environment

Fig. 5. The main GUI of the Society Visualiser

the following constraint (expressed in a generic notation):

$$i : pow(Agent, Act) \Rightarrow i : permitted(Agent, Act)$$

The above constraint states that if at time point i an agent is 'empowered' [19] to perform an action (represented by $pow(Agent, Act)$) then this agent is also 'permitted' to perform the same action (represented by $permitted(Agent, Act)$). The remaining constraints are specified in a similar fashion.

As mentioned earlier, the output of the SV is the global state of the society at a point in time. In other words, given the narrative of time point t (say), the SV will produce the states of affairs that hold at time $t + 1$. Global states are stored in a database and are displayed in a graphical interface for the benefit of the society designer. This graphical display the represents the following information about each agent: (i) institutional powers, (ii) permissions, (iii) obligations, (iv) sanctions and (v) valid actions. Figure 5 demonstrates the GUI of the SV during the simulations of a variation of the CNP (see [5] for more details). Global states are produced for the benefit of the members of the societies as well. Agents can send *query* messages to the SV in order to find out the social state of the group or of their peers. Apart from producing the global state of each time point, the SV includes a number of additional capabilities. One of these capabilities is planning. Given an initial state and a goal state (and the set of social constraints), the SV can determine if there exists a sequence of actions that will lead from the initial to the goal state. Due to the high complexity of such tasks, computation of planning queries can be mainly performed in an off-line phase, that is before the commencement of the interactions of the agent societies. Such a functionality enables the society designer to prove various properties of the agent systems and, therefore, evaluate the design of the social constraints. For example, having specified the constraints of an auction protocol, the designer can determine with the use of planning queries if it is possible to reach a state where two different agents have won the auction. Currently, this functionality is only provided if the social constraints are expressed by means of the $C+$ language[4].

5 Concluding Remarks

We presented a simulation framework that addresses the internal architecture of the agents in addition to accounts of the trust relationships of the agents and of the institutional and social aspects of agent systems. We intentionally did not address/fully investigate issues like low-level details of the simulation framework, experimentation, scalability and other criteria discussed in the community for the experimental study of MAS (see for example [13]). Our aim is to stress the need for the representation of conceptually different perspectives in the modelling and simulation of computational societies. The issues that were not addressed in this paper will be investigated in future work[5].

Several simulation platforms have similar objectives to the framework presented in this paper (e.g. Mace3j [13], `MadKit`, [14], FishMarket [23] and MyWorld [24]). These platforms do not formally address all the issues raised in this paper. For example, in Fishmarket there is a lack of formalisation of concepts such as rights and obligations as well as a lack of an explicit representation of them during simulations. The `MadKit` platform is based on the organisation metaphor AGR (agent/group/role) developed in the context of the Aalaadin project and integrates heterogeneous agent systems. As pointed out in [8], the AGR model views organisations simply as collections of roles and does not incorporate the necessary notion of organisational rules (i.e. social constraints).

There exist several approaches that modify the traditional BDI cycle in order to increase the 'social awareness' of agents (e.g. [25]). Here, we have selected trust as the crucial modifier. The results of our experiments (reported in Section 3) support the notion of trust as an important and beneficial input to agent-based reasoning in MAS. We believe that a similar view can be taken of the normative and institutional concepts discussed in Section 4. We have not fully integrated the trust and deontic concepts of our framework at this stage, but one of our objectives is to make the information compiled by the Society Visualizer available to all agents, either through communication (via *query* messages) or via the addition of a module equivalent to the Society Visualiser to each agent. Reasoning about the institutional and social properties of the agents (as produced by the Society Visualiser) is also a crucial part of trust relationships. Members of a society, governed by a set of rules, must consider when deciding to delegate a task to a peer what are its associated institutional powers, permissions, obligations, sanctions and social roles. For example, it would not be 'rational' to delegate a task to an agent if that agent does not have the institutional power or is forbidden to perform that task. Future work includes modifying the process of the

[4] In this case the SV is an extended version of the `Causal Calculator` software tool (see [6]).

[5] Limited space is another reason why we do not fully investigate these issues here. However, details on experimental results can be found in [5, 6].

trust update mechanism by considering attributes such as powers (institutional and physical), permissions, obligations and roles of the peers.

In conclusion, we believe the combination of architectural, socio-cognitive and external observational perspectives presented in this paper affords a comprehensive approach to modelling MAS. A key aspect of future work will be to refine this combination into a socio-technical substrate defining and supporting the activities of agents *within* MAS as well. We believe that building a socio-technical layer of computation, complementary to (and exploiting functionality of) a lower level of security, is technically feasible, and indeed offers a more tractable solution to certain security problems in e-commerce [26, 27] and digital rights management [28] than, say, ever more sophisticated encryption algorithms or encoding technology. For example, advocates of the Extensible rights Mark-up Language (XrML) position XrML as the foundation of *trusted system development*[29]: a higher-level approach is to use *trust* as the foundation of trusted system development. We plan to fully validate the presented concepts and arguments through continued design, simulation and analysis.

Acknowledgements

This work has been undertaken in the context of the EU-funded ALFEBIITE Project (IST-1999-10298). We have also benefitted from Marek Sergot's contributions on the specification of agent societies and on the development of the Society Visualiser (Section 4). The authors are especially grateful for the comments received both from reviewers and during the *ESAW 2002* workshop.

References

[1] A. Rao and M. Georgeff. BDI agents: from theory to practice. In Victor Lesser, editor, *Proceedings of ICMAS*, pages 312–319, San Francisco, CA, 1995. MIT Press. 53, 55

[2] J. Pitt and A. Mamdani. Designing agent communication languages for multi-agent systems. In F. Garijo and M. Boman, editors, *Proceedings of MAAMAW Workshop*, number 1647 in LNAI, pages 102–114. Springer-Verlag, 1999. 53, 54, 55, 56

[3] C. Castelfranchi and R. Falcone. Social trust: A cognitive approach. In C. Castelfranchi and Y.-H. Tan, editors, *Trust and Deception in Virtual Societies*, pages 55–90. Kluwer Academic Press, 2000. 53, 56, 57

[4] R. Falcone and C. Castelfranchi. The socio-cognitive dynamics of trust: Does trust create trust? In R. Falcone, Y.-H. Tan, and M. Singh, editors, *Trust in Cyber-Societies*, number 2246 in LNAI, 2001. 53, 56

[5] A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In C. Castelfranchi and L. Johnson, editors, *Proceedings of AAMAS*, pages 1053–1062. ACM, 2002. 53, 61, 62, 63, 64

[6] A. Artikis, M. Sergot, and J. Pitt. Specifying electronic societies with the causal calculator. In *Proceedings of AOSE*, pages 75–86, 2002. 53, 61, 64

[7] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *JAAMAS*, 3(3):285–312, 2000. 53, 54

[8] F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):303–328, 2001. 53, 64

[9] J. Filipe. A normative and intentional agent model for organisation modelling. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Proceedings of ESAW*, LNCS. Springer, 2002. 54

[10] A. Ricci, A. Omicini, and E. Denti. Activity theory as a framework for MAS coordination. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Proceedings of ESAW*, LNCS. Springer, 2002. 54

[11] J. Pitt, L. Kamara, and A. Artikis. Interaction patterns and observable commitments in multi-agent trading scenario. In *Proceedings of Autonomous Agents*, pages 481–489. ACM Press, 2001. 54

[12] R. Smith and R. Davis. Distributed problem solving: The contract-net approach. In *Proceedins of the 2nd Conference of Canadian Society for CSI*, 1978. 54, 57

[13] L. Gasser and K. Kakugawa. MACE3J: Fast, flexible distributed simulation of large, large-grain multi-agent systems. In C. Castelfranchi and L. Johnson, editors, *Proceedings of AAMAS*, pages 745–852, 2002. 55, 64

[14] O. Gutknecht, J. Ferber, and F. Michel. Integrating tools and infrastructures for generic multi-agent systems. In *Proceedings of Autonomous agents*, pages 441–448. ACM Press, 2001. 55, 64

[15] M. Witkowski, A. Artikis, and J. Pitt. Experiments in building experiential trust in a society of objective-trust based agents. In R. Falcone, M. Singh, and Y.-H. Tan, editors, *Trust in Cyber Societies*, LNAI 2246, pages 110–132. Springer, 2001. 57

[16] R. Conte. A cognitive memetic analysis of reputation. Alfebiite project deliverable, 2002. http://alfebiite.ee.ic.ac.uk/docs/Deliverables/D5D6.zip. 57, 61

[17] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984. 58

[18] C. Hewitt. Open information systems semantics for distributed artificial intelligence. *Artificial Intelligence*, 47:76–106, 1991. 61

[19] A. Jones and M. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996. 61, 62, 63

[20] M. Sergot. A computational theory of normative positions. *ACM Transactions on Computational Logic*, 2(4):522–581, 2001. 61

[21] M. Shanahan. The event calculus explained. In M. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today*, LNAI 1600, pages 409–430. Springer, 1999. 61

[22] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner. Nonmonotonic causal theories. To appear in the Journal of Artificial Intelligence, 2003. 61

[23] J. Rodriguez-Aguilar, F. Martin, P. Noriega, P. Garcia, and C. Sierra. Towards a test-bed for trading agents in electronic auction markets. In *AI Communications*, pages 5–19, 1998. 64

[24] M. Wooldridge. This is MyWorld: The logic of an agent-oriented DAI testbed. In M. Wooldridge and N. Jennings, editors, *Intelligent Agents: Proceedings of the 1994 Workshop on ATAL*. Springer-Verlag, 1995. 64

[25] J. Broersen, M. Dastani, Z. Huang, J. Hulstijn, and L. Van der Torre. The BOID architecture: Conflicts between beliefs, obligations, intentions and desires. In *Proceedings of Autonomous Agents*, pages 9–16. ACM Press, 2001. 64

[26] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 185–210. Springer, 1999. 65

[27] V. Swarup and J. T. Fabréga. Trust: Benefits, models and mechanisms. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, number 1603 in LNCS, pages 3–18. Springer, 1999. 65

[28] J. Bing. Managing copyright in a digital environment. In I. Butterworth, editor, *The Impact of Electronic Publishing on the Academic Community*, pages 52–62. Portland Press, 1998. 65

[29] XrML. Extensible rights mark-up language. http://www.xrml.org, 2002. 65

Co-Fields: Towards a Unifying Approach to the Engineering of Swarm Intelligent Systems

Marco Mamei[1,2], Franco Zambonelli[2], and Letizia Leonardi[1]

[1] Dipartimento di Ingegneria dellíInformazione
Univ. di Modena e Reggio Emilia
Via Vignolese 905 - 41100 Modena, Italy
[2] Dipartimento di Scienze e Metodi dellíIngegneria
Univ. di Modena e Reggio Emilia
Via Allegri 13 - 42100 Modena, Italy
{mamei.marco,franco.zambonelli}@unimo.it
letizia.leonardi@unimo.it

Abstract. Swarm intelligent systems, in which the paths to problem solving emerge as the result of interactions between simple autonomous components (agents or ants) and between them and their environment, appear very promising to develop robust and flexible software application. However, the variety of swarm-based approaches that have been proposed so far still lacks a common modeling and engineering methodology. In the attempt to overcome this problem, this paper presents a general coordination methodology in which swarmís components are simply driven by abstract computational force fields (*Co-Fields*), generated either by agents, or by the environment. By having agents be driven in their activities by such fields, globally coordinated behaviors can naturally emerge. Although this model still does not offer a complete engineering methodology, it can provide a unifying abstraction for swarm intelligent systems and it can also be exploited to formalize these systems in terms of dynamical systems whose behavior can be described via differential equations. Several example of swarm systems modeled with Co-Fields are presented to support our thesis.

1 Introduction

In the last few years, the research of radically new approaches to software engineering and computing has witnessed a great momentum. These efforts are well justified by the state of trouble of present day computer science. Actual software engineering practices, based on structured design, lead to brittle and fragile systems [20, 21]. While such practices can be acceptable when engineering software systems to be executed in closed and static scenarios, it becomes definitely unsuitable for those systems whose execution has to deal with dynamic and open environments, as it is the

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 68-81, 2003.

case of pervasive [6], mobile [10], and wide-area Internet applications [5]. There, dynamic and openness call for systems architectures capable of supporting both partial failures and dynamic unsupervised re-organization of interaction patterns.

To face this situation, several researchers turned their attention to engineering approaches that take inspiration from the collective behavior of social animals, e.g., ants. From the engineering point of view it is important to observe that animals solve a wide variety of different problems (e.g., finding food, use shortest paths to reach the nest, organize task division) in a very robust and flexible way. Robustness arises from the fact that the colony has the ability to achieve its goal even when some individuals die or fail to perform their task; flexibility arises from the fact that the patterns of interactions between the individuals are not fixed by contract and can be instead dynamically re-shaped to self-adapt to changing environments. Following other authors [4], we refer to those kind of systems as swarm intelligent systems, to stress the fact that their features and capabilities are not embedded in the single components of the system, but emerge by the coordinated activities of a swarm of individuals.

These features of swarm intelligent systems attract more and more software engineering researchers [17], due to the fact that the daily problems solved by groups of animals have direct counterparts in engineering and computer science: finding food strategies can be exploited in an information retrieval context; the strategies used to find the shortest path connecting two locations can be exploited in routing algorithms for telecommunication and computer networks; mechanisms for antís division of labor can be exploited in manufacturing or workflow management scenarios. Therefore, it is no surprise that several works [3, 4, 9, 11, 14] describe peculiar applications of swarm intelligence in a variety of areas. Just to cite a few recent examples, the Anthill system exploits swarm intelligence to perform information search in a wide-area peer-to-peer system [2], while swarm intelligent algorithms have been used to control the collective coordinated behaviors of a swarm of simple robots [8, 19]. While the opportunity to exploit these new concepts has already been spotted, the next challenge is it to enabling their exploitation in a systematic and engineered ways. This is not a simple task and the research community is still far from proposing a whatsoever solution. In our opinion, a pre-requisite to develop such a methodology is to develop a common and general framework in which all the swam intelligent systems could fit. This is exactly the aim of this paper: to present a model able to provide a unifying abstraction (and thus a general framework) for a large class of swarm intelligent systems. We are still far from a true engineering methodology, but at least we have abstractions to deal with different examples of swam intelligent systems in a coherent way.

The model proposed is based on the physically inspired concept of computational fields (Co-Fields). Co-Fields are distributed data structures spread in some space, abstracting some features of the world perceived by the agents, and influencing agentsí actions so as to implicitly encode coordination tasks. The key advantage of Co-Fields is that it enables to represent in a simple yet effective way both different types of ìenvironmentsî in which agents can live and different coordination tasks they have to achieve. This makes it possible to adopt the same basic approach to model different classes of swarm intelligent systems. In addition, the physical inspiration also allows to formally treating Co-Fields coordinated systems in terms of dynamical systems [15, 16] and to derive the differential equations governing their behavior. In the future we think that very interesting results could be obtained by applying classic dynamical system theorems and results to this mathematical description, like for example detecting the systemís attractors and their basin. However, up to now, we used it as a very effective fast prototyping tool: by numerically solving the differential equations it is possible to quickly analyze the behavior of the system, tuning coefficients and make experiments.

The paper is structured as follows. Section 2 describes the Co-Fields model in general. Section 3 presents different classic examples of swarm intelligence and shows how they can all be modeled in terms of the Co-Fields model. Section 4 selects one of the previous examples and shows how it is possible to describe it using a dynamical systemsí formalism. Section 5 concludes the paper and presents our future work.

2 The Co-Fields Model

The Co-Fields model provides a modeling framework to design a multi agent system (MAS). Its primary focus is to consider and model the MAS as a ìwholeî, in which agents achieve their goal not because of their single capabilities, but because they are part of an self-organized system that leads them to the goal achievement [17]; the fact that the goals are accomplished is not a merit of the single agents, but of the system as a whole.

The Co-Fields model can be schematized in the following four points:

1. The environment is represented and abstracted by fields, spread by agents and by the environment itself. These fields convey some useful information for the agentsí coordination tasks and provide agents with strong coordination-task-tailored context awareness.
2. The coordination policy is realized by letting the agents to move following the ìwaveformî of these fields.
3. Environment dynamics and agentsí movements induce changes in the fieldsí surface, composing a feedback cycle that influences agentsí movement (point 2).
4. This feedback cycle lets the system (agents and environment) to auto-organize, so that the coordination task is finally achieved.

More in detail, a field can be defined as a distributed data structure composed by a unique identifier, a value (representing the field magnitude in that particular point), and a propagation rule. Fields can be generated by the agents or by the environment, and are propagated through the *space* as specified by their propagation rule, which thus determines the final shape of the field surface. Fields can be static or dynamic, basically a field is static if once propagated its magnitude does not change over time; it is dynamic if its magnitude does. A field can be dynamic because for example its source moves and the field, with some propagation delay, changes accordingly its magnitude, or because for example its propagation rule it is designed to remain active and to change field value after some time. In our model agents simply compute a sort of combination of the fields they perceive (e.g. a linear combination). The result of this combination is itself a field, that we will call the agentís *coordination field*.

Agents are forced to follow (deterministically or with some probability) the shape of their *coordination field*, like if they were walking upon the *coordination field* associated surface. Basically their actions will be based on following downhill the decrease of the *coordination field*, on following its increase uphill, or on following one of its equipotential lines (see Figure 1).

Complex movements are achieved not because of the agent will, but because of dynamic re-shaping of this surface. This is a strong point that will enable us to completely treat these field-based software systems as dynamical systems. In fact, having an analytical description of the coordination field, as will be shown in section 4, is rather easy to write down a set of differential equations that can mathematically express the bond between the agentsí movement and the gradient of the fieldís surface, and thus govern and describe the systemís behavior.

Almost all the coordination tasks considered in this paper deal with agentsí movements and so the *coordination field* can be considered spread in the physical space and following the *coordination field* surface means moving from one place to another. However, the approach is more general and a *coordination field* spread in a more abstract space could encode coordination tasks that do not deal with some physical movements, but with other kind of actions. In these cases agents would follow its *coordination field*, not by moving from one place to another, but by making other kind of actions. An example of this more abstract encoding will be presented at the end of the next section.

3 Classic Swarm Intelligence Examples

In this section we are going to briefly survey different swarm intelligent coordination policies, mainly taken from [3, 11, 14] and we will show how can they be modeled in terms of the Co-Fields model. This will serve both to justify the claim of unifying approach of this paper and to clarify the Co-Fields model itself.

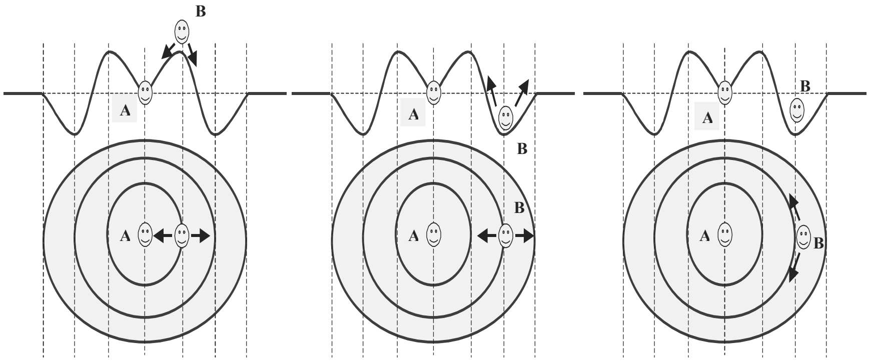


Fig 1. Agent B following the surface of its *coordination field* which coincides with the field generated by agent A; (**left**) the agent B follows downhill the decrease of its *coordination field*; (**center**) the agent B follows uphill the increase of its *coordination field*; (**right**) the agent B follows an equipotential line of its *coordination field*

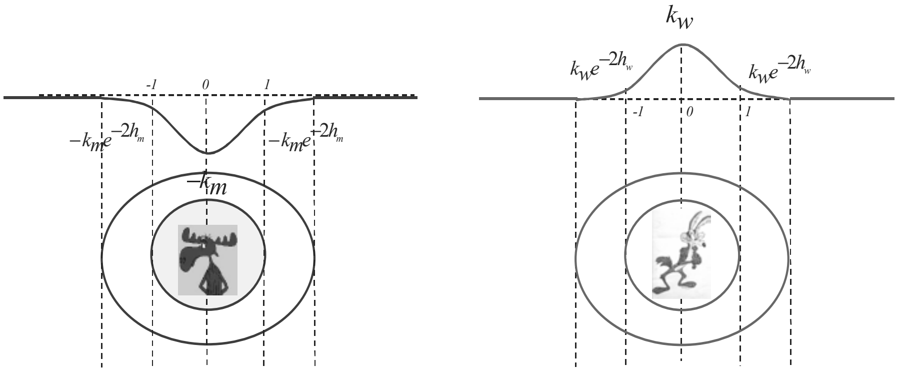


Fig. 2. Mooseís (**left**) and wolfís (**right**) generated field

3.1 Wolves: Surrounding a Prey

To capture a moose, a pack of wolves have to act in a coordinated way, surrounding the prey. Their coordinated behavior can be explained by means of swarm intelligence without assuming long-range communication mechanisms or complex intelligent strategies [11, 14]. Wolves simply hunt for the moose trying to maintain a suitable distance from other wolves. Simulations of this simple strategy, with a moose that simply try to escape by moving farthest away from the nearest wolf, shows that it is a viable and successful solution for wolves to surround and to capture the prey.

To model such behavior with Co-Field, we can imagine that the moose and the wolves generates the following fields, to be updated in real-time depending on agentsí movements: a moose at the coordinates (X_m, Y_m), generates a *mooseís field* described by the following equation (see Figure 2 left):

$$moose(x,y,t) = -k_m e^{-h_m\left((x-X_m)^2+(y-Y_m)^2\right)} \qquad k_m, h_m > 0$$

Analogously, the wolf i at the coordinates (X_w^i, Y_w^i), generates a *wolfís field* whose equation is (see Figure 2 right):

$$wolf_i(x,y,t) = k_w e^{-h_w\left((x-X_w^i)^2+(y-Y_w^i)^2\right)} \qquad k_w, h_w > 0$$

The contribution of this analytical description will be clear in section 4, when we will derive the differential equations describing the Co-Fields model in general, and we will apply those equations to this mathematical description of the fieldsí surfaces to derive mooseís and wolvesí behavior.

Now, if we consider the mooseís *coordination field* consisting in the linear combination of all the wolvesí fields:

$$coord_{moose}(x,y,t) = \overset{n}{\underset{i=1}{!}}\ wolf_i(x,y,t)$$

by following the decrease of the resulting field, the moose runs away from wolves. Similarly, if we consider each wolfís *coordination field* consisting in the linear combination between the mooseís field and all the other wolvesí fields:

$$coord_{wolf_i}(x,y,t) = moose(x,y,t) + \overset{n}{\underset{j=1,j\neq i}{!}} wolf_j(x,y,t)$$

a wolf is directed towards the moose, but staying away from other wolves. It is clear that this simple description is perfectly analogous to the description based on distances and, by a proper tuning of the fieldsí coefficients, can lead to the same results.

3.2 Birds Flocking

Flocks of birds stay together, coordinate turns, and avoid each other, by following a very simple swarm algorithm [11, 14]. Similar problems happen in air-traffic control and convoys of ships and they could be possibly addressed by using similar methods. The coordinated behavior of flocks can be explained by assuming that each bird tries to maintain a specified separation from the nearest birds and to match nearby birdsë velocity. The flock is a self-constraining structure in which each entityís individual action simultaneously responds to and changes the overall structure of the flock. Although each bird senses only the movements of its nearest peers, its responses to these movements propagate to others, so that the system as a whole exhibits global coordination.

To model this strategy under the Co-Fields modeling framework, we can imagine that each bird generates a *FLOCK* field like the one showed in figure 3 (left), which can be described by the following simple equation:

$$d = \sqrt{(x - X_P^i)^2 + (y - Y_P^i)^2}$$

$$FLOCK_i(x,y,t) = d^4 - 2a^2 \cdot d^2$$

This field is updated in real time to match the birdís movements. Now, saying that birds have to stay a specified distances from each other is equivalent to saying that birds has to follow the decrease of other birdsí generated fields (see Figure 3 right). In fact the shape of the field constrains birds to stay close each other in an almost regular grid, the movement of a bird and the consequent change in the field it generates force other birds to move as well.

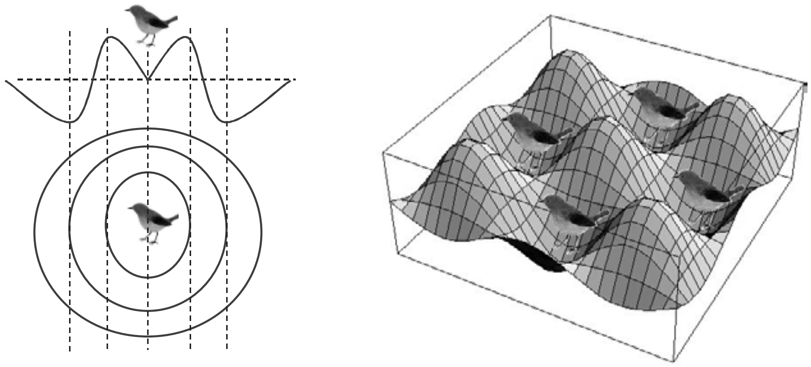


Fig. 3. Flocking field (**left**) flocking birds (**right**)

3.3 Ant Foraging

Ants construct a network of paths that connects their nest with available food sources [3, 14]. This global structure emerges from the simple actions of the individual ants that do not even know they are involved in this distributed building process. Several authors spotted the possibility to exploit the technique used by ants in this process in a routing or in an information retrieval context. Basically each ant that forages for food is able to produce two kinds of pheromones (scent markers that many insects generate). The home-pheromone is produced after leaving the anthill wandering looking for food. The food-pheromone is produced when the ant goes back to the anthill after some food has been found. Ants wander randomly following the food pheromone when looking for food, following the home pheromone when bringing food back to the anthill. The overall system behavior is based on the fact that, pheromone tracks deployed by an ant can be exploited by the same or by other ants in the future. The natural tendency of the pheromones to evaporate if not reinforced, allows the pheromone network to remain up-to-dated. So when a food source is extinguished the corresponding pheromone trial disappears, because it is no longer used and reinforced.

To describe the antsí foraging strategy under the Co-Fields model, we can imagine that the environment is abstracted (i.e. perceived by ants) by means of two (initially flat) fields. These two fields, which we will call the home and the food fields, are generated and spread by the environment itself. The environment reacts to antsí movements by wrinkling the fieldsí surface, while antsí movements are affected by the ìwaveformî of the field. This feedback cycle constitutes the key to let the system auto-organize. The algorithm followed by ants can in fact be restated by supposing that each ant wanders, avoiding obstacles, following (probabilistically) the decrease of the food field when it is looking for food, following (probabilistically) the decrease of the home field when it is bringing food back to the anthill. Then to close successfully the feedback cycle we will simply imagine that the environment reacts to the antsí presence, by locally wrinkling the home field surface in correspondence of the points in which ants looking for food are located and it wrinkles the food field surface in the points in which ants carrying food are located. The form of the wrinkle is depicted in figure 4 (left). Moreover we will suppose that the environment is able to control the deepness of the wrinkle, so that each new (or renewed) wrinkle is deeper that all the wrinkles in its neighborhood. Following these principles, antsí movement creates a network of channels in the fieldsí surfaces. In the food-field surface these channels descend from the anthill to the food sources, in the home-field surface these channels descend from the food sources to the anthill. In figure 4 (right) a channel created by an ant movement is shown. Of course this is a pure abstraction, in no way a natural environment can provide the described capabilities. However this does not matter for our purposes, in fact we just want to demonstrate that the Co-Fields model can abstract these phenomena, and then use this model to build software systems in an artificial environment, where these functionalities can easily be gathered.

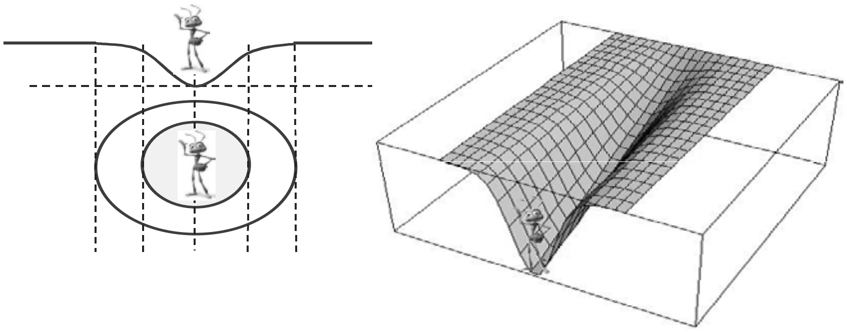


Fig. 4. A wrinkle induced on a fieldís surface by an ant presence (**left**). The channel in the field surface created by the ant movement (**right**)

Finally, to accustom for pheromone evaporation, we will imagine that each of the fieldsí surfaces has some form of memory and it reshapes to its original flat form if untouched. In particular this can be formally stated by saying that a fieldís wrinkle at coordinates (X_W, Y_W) created a time t_0 can be mathematically described by the following function:

$$wrinkle(x, y, t) = -k(t - t_0)e^{-h\left((x-X_w)^2+(y-Y_w)^2\right)} \qquad k(t), h > 0$$

Where *k(t)* is a function that goes to zero as *t* increases, while *h* is a constant that expresses the fact the wrinkle is only a local deformation.

In general this kind of analytical description is not only useful to state these concepts formally, but it can be applied to provide a dynamical system description of the model. In section 4 this procedure will be applied to the example discussed in section 3.3, however the results can be easily applied also to this case.

3.4 Ant Labor Division and Task Succession

In social insects, different activities are often performed simultaneously by specialized individuals; this phenomenon is called division of labor. A key feature of division of labor is plasticity: the ratios of workers performing different tasks can vary in response to internal perturbations or external challenges. A simple model, which relies on response threshold, can explain how ants achieve flexibility and specialization in labor division [3, 11]. Each individual has a response threshold for every task, it engages in task performance when the level of the task-associated stimuli exceeds its threshold, it drops a task when the task associated stimuli falls under another threshold. In this way everyone adjust its duties according to the colonyís needs. Moreover, by performing a certain task individualsí task associated threshold decrease. This simple strategy is the key for specialization: the more one individual performed a task in the past, the more likely the same individual will perform the task in the future.

This example is particularly interesting to be modeled with Co-Fields, because it involves fields propagated in a space, which is not the physical space in which ants are embedded.

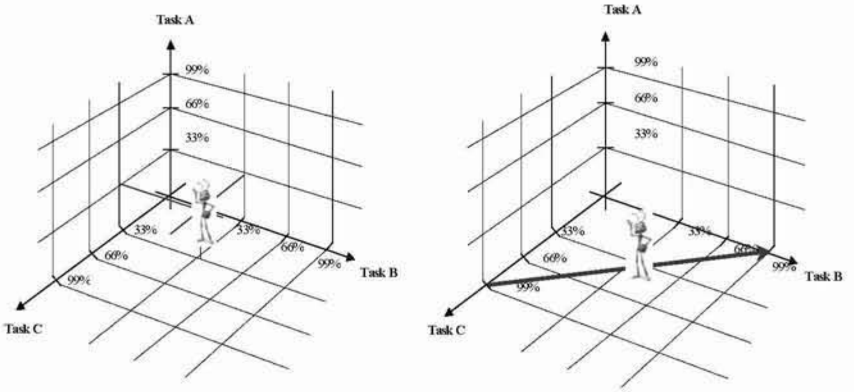


Fig. 5. Diagram to represent task space (**left**). Movement in the task space (**right**)

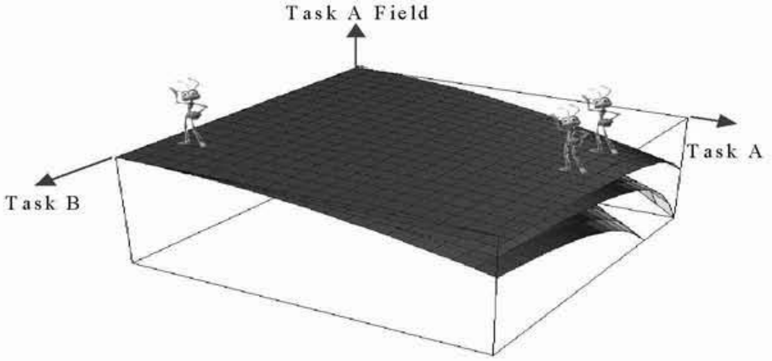


Fig. 6. Task-A stimuli associated fields. The more the Task A's achievement becomes urgent, the more the overall surface gets steep

To model the above described coordination task within the Co-Fields approach, we can imagine that there exists a virtual space, separated by the physical one, in which ants are embedded. This is a multi-dimensional space containing one dimension for each of the possible tasks an ant may be involved. If an ant can be involved just in three tasks, let's say: Task A, Task B, Task C, then this space will be the one depicted in figure 5 (left). An ant in placed within the space depending on its duties: we can suppose that each axis measures the fraction of the time, an ant performs that particular task. So that the ant in figure 5 (left) performs Task A for the 33% of its time, Task B for the 33% of its time and Task C for the 33% of its time. Of course, in general, every ant is constrained to the subspace $\sum_{x_i \in Tasks} x_i \leq 100\%$. An ant can move in this space, but its movement does not correspond to an actual movement in its physical space, but in a change in the ant's duties. So for example the movement depicted in figure 5 (right), represent an agent that gradually stops doing Task C and starts doing Task B. Of course fields can be spread and sensed by ants also in this space. In particular the environment generates fields encoding the stimuli that encourages ants in performing a task. Basically the shape of these fields is almost like a steep flat surface, decreasing towards the direction of the associated axis. The more the task achievement is urgent, the more the environment increases the steepness of the field surface. In figure 6, three different Task Aís associated fields are shown. The more the surface get steep, the more Task Aís achievement becomes urgent.

Antsí algorithm can thus be restated by means of the following actions: each ant evaluates a combination of the sensed fields, by considering only those fields whose steepness where the ant is located overcomes a certain threshold. Then the ant follows downhill the field obtained. As the ants move along the Task space the fieldís surface tend to get flattened, because the associated tasks are achieved. Antsí duties get stabilized in a suitable configuration, until new stimuli and thus fieldsí surfacesí deformations appear. As depicted in figure 6, a task-associated fieldís surface is not actually a steep flat surface, but its steepness increases both in the direction of the associated axis and both in the proximity of the associated axis itself. The reason for this non-linear shape is to enforce specialization. In fact, on the one hand by increasing the steepness towards the direction of the associated axis, the more an ant is placed towards that direction (i.e. the more it is performing that task), the more it will easy that the ant will perform the task in the future because the fieldís steepness is there particularly high. On the other hand the increase of the steepness towards the axis itself (i.e. towards the zero) rends easier that ìunemployedî ants, rather than already fully committed ants, engage the task.

4 A Dynamical Description of Swarm Intelligence

In this section we are going to present a dynamical system description of the Co-Fields model, based on the differential equations that govern the system behavior.

The Co-Fields model suits really well to this kind of description, because individualsí behavior can be assimilated to the one of a ball rolling on a surface (the *coordination field* surface). It is thus rather easy to see that if we consider the individual *i*, and we denote its coordinates in a particular space as ($x_1^i(t), x_2^i(t),..., x_n^i(t)$) and its *coordination field* as $coord_i(X_1, X_2,..., X_n, t)$. Then the differential equations that govern *i* behavior are in the form:

$$\frac{dx_j^i}{dt} = \pm v \cdot \frac{\partial coord_i(X_1, X_2,..., X_n, t)}{\partial X_j} \qquad j = 1,2,..., n$$

Where *v* expresses the ìspeedî of individual *i*, while the sign at the right of the equals is decided by weather *i* follows the increase or the decrease of the coordination field. This is because the direction of the gradient of the individualís *coordination field*, evaluated towards the spatial coordinates, point to the direction in which the *coordination field* increase. So the individual *i* will follow this gradient or will go in the opposite direction depending on if it wants to follow the increase or the decrease of its *coordination field*. In a similar way it is possible to model those cases in which *i* follows the equipotential lines of the fields.

It is worth noting that the above description is completely deterministic, while most of the swarm intelligent systems have random decision process at their core (e.g.

foraging ants follow pheromone trails only with some probability). We think that this mismatch is not relevant for the claim made in the paper that Co-Fields can provide a unifying abstraction for swarm intelligent systems. In fact, random elements are not explicitly necessary to generate swarm systemsí collective coordinated behaviors and most of the swarm intelligence is maintained even if randomness is removed. Despite of this, when developing swarm intelligent software applications, random elements can be very useful to give robustness and flexibility to the system and we plan to introduce them in our model as described in the future works section.

To clarify the above equations and to describe a concrete example, we are going to write the differential equations that govern the moose escape and the wolves surrounding strategy. The moose coordination field, following section 3.1, is given by the sum (linear combination with all the coefficients equal to 1) of the wolvesí fields (letís suppose n wolves).

$$coord_{moose}(x,y,t) = \sum_{i=1}^{n} wolf_i(x,y,t)$$

While the wolf *i coordination field* is given by the sum (linear combination with all the coefficients equal to 1) of the mooseís field (that attracts the wolf) and all the others wolvesí fields (that repulse the wolf).

$$coord_{wolf_i}(x,y,t) = moose(x,y,t) + \sum_{j=1, j\neq i}^{n} wolf_j(x,y,t)$$

Now writing the differential equations is just a matter of substituting these *coordination fields* with the fieldsí equations described in 3.3 and to use the differential equations at the beginning of this section, in the case of individuals following the decrease of the *coordination field*. Because of space limit, we will not report the result of this substitution, however the system turns out to be described by a set of strictly coupled non-linear differential equations (because of the exponentials in the fieldsí functions in 3.3). Obviously, it is not easy to solve these differential equations analytically, but it is straightforward to obtain a numerical solution with any math software toolkit e.g. [13].

In general, numerical integration of the equations describing the system can be regarded as a very effective fast prototyping tool to quickly analyze the behavior of the system, tuning coefficients and make experiments. Moreover, in the future we think that very interesting results could be obtained by applying classic dynamical system theorems and results (that do not need the equationsí analytic solution) to this mathematical description, like for example detecting the systemís attractors and their basin [18].

The results of a numerical integration of the exampleís differential equations, in the case of a system composed by one moose and three wolves, are displayed in the following figure (see Figure 7), which shows a common x-y plane with the trajectories of the elements of the system (i.e. the solutions of $(x_i(t), y_i(t))$ evaluated for a certain time interval). It is worth noting that solving these equations numerically is a very effective way to simulate the system, and it can be regarded as a very easy and powerful tool to make experiments and tuning coefficients.

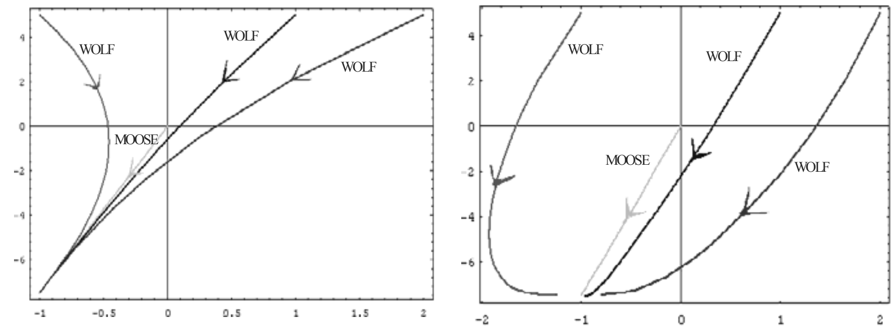


Fig. 7. (**left**) Wolves do not care about other wolvesí fields and thus they are not able to surround the moose. (**right**) Wolves sensing other wolvesí fields are able to surround the moose

5 Conclusions, Current and Future Works

In the long run to develop a systematic and engineered methodology to deal with swarm intelligent system, in our opinion, a pre-requisite is to develop a common and general framework in which all the swam intelligent systems could fit. In this paper, we presented a model able to provide a unifying abstraction (and thus a general framework) for a large class of swarm intelligent systems. We are still far from a true engineering methodology, but at least we have abstractions to deal with different examples of swam intelligent systems in a coherent way. This abstraction has been also exploited to formalize some of these systems in terms of dynamical systems whose behavior is described by means of differential equations. The numerical solutions of these equations provide a valuable tool to make experiments on the studied systems.

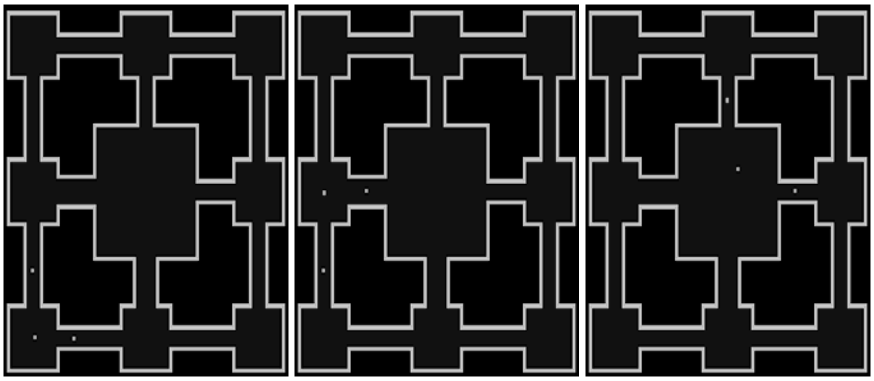


Fig. 8. (**from left to right**). Different stages in the Co-Fields guided movement of a group of agents through a virtual building. Agents move, while remaining in a flock

In our current work we are applying the Co-Fields model to concrete software systems. In particular, we are dealing with those applications in which a group of autonomous components (i.e. agents), running on some mobile computing devices, have to coordinate their respective movements (or the movements of the users carrying them on). The goals of this coordination can be various: letting the agents to meet somewhere, to move avoiding traffic jams, to distribute themselves accordingly to a particular geometry, etc. The first results of this work have been the development of a simulation in which a field-based application guides the visitors and the clerks of a museum towards the museumís building [12]. In Figure 8, we sketch the results of such a simulation when applied to the flocking problem, and visualizing the coordinated movements of a set of security guards within the museum.

We also emphasize that in Co-Fields, as in other swarm systems, adaptivity and self-organization is guaranteed: without any change to the agentsí code, the security guards can continue move in a coordinated way even when new security guards arrive or when security guards have to take care of a totally different museum. With regard to this argument, we plan to extend the Co-Fields model with random elements in agentsí behavior (random elements are present in most swarm intelligent system to add robustness and flexibility). These random elements could be introduced either by assuming that agents behavior is not deterministic, but random (i.e. the agent follows downhill the fieldís surface only with some probability), or by assuming that random noise is added to the fields propagated in the environment, resulting in a fluctuating field surface that the agents follow deterministically.

Future work will proceed towards two main directions: on the one hand we will try to extend the Co-Fields model and to better formalize it. Our perception is that particularly significant results can be obtained by generalizing the use of the Co-Fields model to tasks that do not deal with some physical movements, but with other kind of actions, as we did in the example of antís division of labor. In this direction, we will explore other approaches like Amorphous Computing [1], Dissipative Cellular Automata DCA [7] and Artificial Neural Networks [9] to see if and how the Co-Fields model can be combined with them.

On the other hand, we would like to develop a proper infrastructure middleware to support the Co-Fields model in a real computing scenario. The role of this infrastructure would be to manage the distributed data structures associated to fields by letting agents to generate and sense fields.

References

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman and R. Weiss, ìAmorphous Computingî, Communications of the ACM, 43(5), May 2000.

[2] O. Babaoglu, H. Meling, A. Montresor, ìAnthill: A Framework for the Development of Agent-Based Peer-to-Peer Systemsî, 22nd International Conference on Distributed Computing Systems (ICDCS), Vienna (A), July 2002.

[3] E. Bonabeau, M. Dorigo, G. Theraulaz, ìSwarm Intelligenceî, Oxford University Press, 1999.

[4] E. Bonabeau, G. Theraulaz, ìSwarm smartsî, Scientific American: 72-79, March 2000.

[5] G. Cabri, L. Leonardi, F. Zambonelli, ìEngineering Mobile Agent Applications via Context-Dependent Coordinationî, IEEE Transactions on Software Engineering, 28(9), September 2002.

[6] D. Estrin, D. Culler, K. Pister, G. Sukjatme, ìConnecting the Physical World with Pervasive Networksî, IEEE Pervasive Computing, 1(1):59-69, Jan. 2002.

[7] M. Gatti, M. Spallanzani, A. Roli, F. Zambonelli, ìWhat Can Cellular Automata Tell Us About the Behaviour of Large Multi-Agent Systems?î, 1st International Workshop on Software Engineering for Large Agent Systems, Orlando (FL), May 2002.

[8] S. Johansson, J. Johansson, A. Saffiotti, ìAn electric field approach to autonomous robot controlî, Technical Report 2000:17, Blekinge Institute of Technology, Dec. 2000.

[9] S. Johnson, ìEmergenceî, Scribner, 2001.

[10] J. M Kahn, R. H. Katz, K. S. J. Pister, ìEmerging Challenges: Mobile Networking for Smart Dustî, Journal of Communications and Networks, 2(3):188-196, Sept. 2000.

[11] J. Kennedy, R.C. Eberhart, Y. Shi, ìSwarm Intelligenceî, Morgan Kaufmann, 2001.

[12] M. Mamei, L. Leonardi, M. Mahan, F. Zambonelli, ìCoordinating Mobility in a Pervasive Computing Scenario with Co-Fieldsî, 1st International Workshop on Mobile Teamwork (at ICDCS 02), IEEE CS Press, Vienna (A), July 2002.

[13] Mathematica, http://www.wolfram.com/products/mathematica.

[14] H.V.D. Parunak, ìGo to the Ant: Engineering Principles from Natural Multi-Agent Systemsî, Annals of Operations Research 75:69-101, 1997.

[15] H.V.D. Parunak, ìA Dynamical Systems Perspective on Agent-Based Going Concernsî, IWMAS, Dedham, MA, USA, 1998.

[16] H.V.D. Parunak, ìEntropy and Self-Organization in Multi Agent Systemsî, in Proceedings of Autonomous Agents, Montreal, Canada, May 2001.

[17] H. V. Parunak, S. Brueckner, J Sauter, ìERIMís Approach to Fine-Grained Agentsî, NASA Workshop on Radical Agent Concepts, Greenbelt, MD, USA, Jan. 2002.

[18] E. Rauch, M. Millonas, D. Chialvo, ìPattern Formation and Functionality in Swarm Modelsî, Physics Letters A 207 pp. 185-193, 1995.

[19] M. Yim, Y. Zhang, D. Duff, ìModular Robotsî, IEEE Spectrum, Feb. 2002.

[20] G. Sussman, ìRobust Design through Diversityî, DARPA/MIT Amorphous Computing Workshop, September 1999.

[21] F. Zambonelli, V. Parunak, ìSign of a Revolution in Computer Science and Software Engineeringî, in this volume.

A Schema for Specifying Computational Autonomy

Matthias Nickles, Michael Rovatsos, and Gerhard Weiß

Institut für Informatik, Technische Universität München,
D-85748 Garching, Germany
{nickles,rovatsos,weissg}@in.tum.de

Abstract. A key property associated with computational agency is autonomy, and it is broadly agreed that agents as autonomous entities (or autonomous software in general) have the capacity to become an enabling technology for a variety of complex applications in fields such as telecommunications, e/m-commerce, and pervasive computing. This raises the strong need for techniques that support developers of agent-oriented applications in specifying the kind and level of autonomy they want to ascribe to the individual agents. This paper describes a specification schema called RNS ("Roles, Norms, Sanctions") that has been developed in response to this need. The basic view underlying RNS is that agents act as owners of roles in order to attain their individual and joint goals. As a role owner an agent is exposed to certain norms (permissions, obligations and interdictions), and through behaving in conformity with or in deviation from norms an agent becomes exposed to certain sanctions (reward and punishment). RNS has several desirable features which together make it unique and distinct from other approaches to autonomy specification. In particular, unlike other approaches RNS is strongly expressive and makes it possible to specify autonomy at a very precise level. Moreover, RNS is domain- and application-independent, and is of particular value for agent-oriented requirements elicitation and analysis.

1 Introduction

A key property associated with computational agency is autonomy. As an autonomous entity, an agent possesses action choice and is able to act under self-control within the bounds of its design objectives. Compared to other basic properties usually associated with agency such as behavioral flexibility (covering both reactivity and pro-activeness) and high-level interactivity (based on communication and negotiation with the purpose of cooperation or competition), it is autonomy that makes agent orientation most distinct from traditional software and systems development paradigms. These paradigms, relying on concepts such as objects or components, simply are not intended to capture the notion of computational autonomy – to the contrary, they can even be said to be intended to prevent autonomous component behavior. The past years have witnessed a rapidly growing interest in various aspects of computational autonomy, as it is also indicated by an increasing number of related research efforts

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 82–95, 2003.

(e.g., see [1, 4, 5, 8] for work explicitly dealing with autonomy). This interest is largely based on the insight that agents as autonomous entities – or autonomous software in general – do have the capacity to become an enabling technology for a broad and important class of applications, namely, applications that run in complex socio-technical environments which are open, dynamic, networked, time-critical, and/or decentralized and thus possess a critical mass of inherent unpredictability. Due to this unpredictability a full behavioral specification of application software is not possible in many cases, and this is the point where autonomously acting entities come into play which are able to act in a desired manner not only in anticipated environmental situations but also in situations that were unforeseeable at design time. Well known examples of application domains calling for a deployment of "autonomous technology" are telecommunications, logistics, e/m-commerce, supply chain management, and pervasive and ubiquitous computing.

Among the most critical activities in engineering agent-based applications is the specification of the kind and level of autonomy owned by the different agents. This specification can fail in two opposite ways, both making it unlikely that a resulting application meets its requirements: on the one hand, if this specification is too rigid then necessary action choice is suppressed and "objects are made out of agents"; and on the other hand, if this specification is too generous then unnecessary action choice is admitted and "agents are made out of objects". This *autonomy specification dilemma* raises the strong need for supporting a developer in specifying the agents' action choice. More specifically, what is needed are specification techniques – methods, formalisms, tools, languages, and so forth – which enable and force developers to *precisely* state what degrees of behavioral freedom they want to ascribe to the individual agents. This paper describes a specification schema called RNS (standing for "Roles, Norms, Sanctions") which has been developed in response to this need. This schema employs the concepts of roles, norms (permissions, obligations, and interdictions) and sanctions (reward and punishment) to capture autonomy. RNS, which is particularly suited for agent-oriented requirements elicitation and analysis, possesses several desirable features which together make it unique and distinct from related approaches to autonomy specification. In particular, although RNS is based on a relatively simple and easy-to-understand notation and syntax, it is very expressive and enables a developer to specify agent autonomy with a very high precision. Moreover, RNS is general enough to be not bound to any specific application or application domain.

The paper is structured as follows. Section 2 describes RNS in detail; this includes a general characterization (2.1) and a detailed technical presentation (2.2 and 2.3). This section also gives a number of illustrating examples of all basic aspects of RNS. Finally, Section 3 discusses pros and cons of RNS and compares RNS to related approaches.

2 The RNS Schema

2.1 Informal Characterization

RNS employs the concepts of role, norm and sanction as known from sociological role theory (e.g., [3]) to specify autonomy. The basic view underlying RNS is that autonomous agents are embedded in a social frame which regulates – guides and constrains – their behavior. This social frame, called role space, is composed of roles which are available to the agents and through which the agents can try to achieve individual and joint objectives. An agent may own several roles at the same time, and the set of roles owned by an agent may dynamically vary over time. Conceptually roles are viewed as a means for specifying desired behavior and for achieving behavioral predictability, and *not* as a means for making sure that agents do never exhibit unexpected and undesirable behavior. Roles in RNS are not intended to fully constrain individual behavior; instead, they leave room for individuality (agents may fill a role differently by putting emphasis on different aspects). Somewhat more specifically, according to RNS a role consists of at least one activity to which norms together with sanctions are attached. RNS distinguishes three types of norms (permissions, obligations, and interdictions) and two types of sanctions (reward and punishment). Whereas norms specify behavior expectations held by agents against other agents (in their capacity as role owners), sanctions specify potential consequences of norm-conforming and norm-deviating behavior. Sanctions, in some sense, serve as a means for controlling autonomy. By enabling a designer to explicitly specify sanctions, RNS takes care of the fact that generally (and especially in open applications) agents as autonomous entities do not necessarily behave in conformity with available norms, but may also ignore and violate them (be it intentionally or not).

2.2 Basic Concepts and Constructs

The RNS schema requires to analyze and specify systems in terms of roles which are available to the agents and through which the agents can try to achieve their objectives. The set of available roles is called a *role space*. A role space is specified in the form

`ROLE SPACE` *role_space_id* { *role_id_list* }

where *role_space_id* is a character string uniquely identifying the role space under consideration and *role_id_list* is a list of character strings called role identifiers that uniquely identify roles.[1] *Roles* are viewed as collections of specific activities, and for each role identifier, *role_id*, there must be a role specification in the form

`ROLE` *role_id* { *activity_spec_list* }

[1] Syntactic keywords are written in underlined `TYPEWRITER FONT`, and *italic font* is used to indicate variables. Expressions enclosed in brackets [.] are optional. Brackets of the form <.> are part of the RNS syntax. As the reader will see in the examples provided below, most of the variables also can be instantiated with specific keywords such as `EACH`.

where *activity_spec_list* is a list of specifications of all activities being part of the role *role_id*. How activity specifications look like is described in section 2.3.

The RNS schema distinguishes three types of norms – permissions (**P**), obligations (**O**), and interdictions (**I**) – and two types of sanctions – reward (**RE**) and punishment (**PU**) – that apply in case of norm conformity and deviation. Based on these distinctions, a *status range* is attached to each activity which describes activity-specific norms and associated sanctions. More specifically, a status range specification is of the form

$$\underline{\texttt{STATUS RANGE}}\ \textit{status\_statement\_list}$$

where *status_statement_list* is a list of so called *status statements* each describing a norm-sanction pair that is specific to the activity to which the status range is attached. A key feature of the RNS schema is that it facilitates the explicit modeling and specification of *requests* for (refraining from) executing particular activities. This feature induces the distinction of two kinds of norm-sanction pairs attached to an activity:

- norm-sanction pairs an activity is subject to, no matter whether the execution or omission of the activity is requested or not by some agent. Norm-sanction pairs of this kind are, so to say, independent of any requests for (not) executing the activity to which they are attached. Norm-sanction pairs of this kind, and the status statements describing them, are called *independent* and are indicated by the keyword IND.
- norm-sanction pairs an activity becomes subject to as a consequence of a request for (not) executing it. Such norm-sanction pairs are, so to say, induced by (i.e., do become active as an effect of) explicit requests for activity execution or omission. Agents requesting the (non-)execution of an activity are called role senders. Norm-sanction pairs of this kind, and the status statements describing them, are called *dependent* and are indicated by the keyword DEP.

The common syntax of independent status (IS) statements and dependent status (DS) statements is as follows:

$$\langle \textit{status\_type} \rangle : \underbrace{\underbrace{\underline{\texttt{NORM}}\ \langle \textit{norm\_type} \rangle\ \langle \textit{condition} \rangle}_{\text{norm specification}} + \underbrace{\underline{\texttt{SANC}}\ \langle \textit{sanction\_type} \rangle\ \langle \textit{sanction} \rangle}_{\text{sanction specification}}}_{\text{norm-sanction pair}}$$

where *status_type* $\in$ {IND, DEP *role_id*} discriminates among IS and DS statements, *norm_type* $\in \{\mathbf{P},\ \mathbf{O},\ \mathbf{I}\}$, *condition* is a Boolean expression making it possible to formulate conditioned norms, *sanction_type* $\in \{\mathbf{RE},\ \mathbf{PU}\}$, and *sanction* is an expression specifying a sanction of type *sanction_type*. Though syntactically almost identical, IS and DS statements differ significantly in their semantics. First consider IS statements, that is, statements of the form

$$\langle \underline{\texttt{IND}} \rangle : \underline{\texttt{NORM}}\ \langle \textit{norm\_type} \rangle\ \langle \textit{condition} \rangle + \underline{\texttt{SANC}}\ \langle \textit{sanction\_type} \rangle\ \langle \textit{sanction} \rangle$$

Dependent on *norm_type*, such a statement attached to an activity reads as follows:

- *norm_type* = {**P**}: "An agent owning the role of which this activity is part of is *permitted* to execute this activity provided that the condition *condition* is fulfilled. The sanction associated with this permission is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**O**}: "An agent owning the role of which this activity is part of is *obliged* to execute this activity provided that the condition *condition* is fulfilled. The sanction associated with this obligation is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**I**}: "An agent owning the role of which this activity is part of is *forbidden* to execute this activity provided that the condition *condition* is fulfilled. The sanction associated with this interdiction is of type *sanction_type* and is given by *sanction*."

Against that, DS statements, that is, statements of the form

<DEP *role_id*> : NORM <*norm_type*> <*condition*> + SANC <*sanction_type*> <*sanction*>

read as follows:

- *norm_type* = {**P**}: "If an agent owning the role *role_id* requests to execute this activity (from an agent owning the role of which this activity is part of), then the requested agent is *permitted* (by the requesting agent) to execute it (i.e., she may execute it) provided that the condition *condition* is fulfilled. The sanction associated with this permission is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**O**}: "If an agent owning the role *role_id* requests to execute this activity, then the requested agent is *obliged* (by the requesting agent) to execute it (i.e., she must execute it) provided that the condition *condition* is fulfilled. The sanction associated with this obligation is of type *sanction_type* and is given by *sanction*."
- *norm_type* = {**I**}: "If an agent owning the role *role_id* requests to *not* execute this activity, then the requested agent is *forbidden* (by the requesting agent) to execute it (i.e., she must not execute it) provided that the condition *condition* is fulfilled. The sanction associated with this interdiction is of type *sanction_type* and is given by *sanction*."

DS statements make it possible to capture situations in which requests (e.g., from different agents) for executing an activity do have different normative and sanctioning impacts on the requested agent. In other words, DS statements allow to model situations in which requests even for the very same activity induce different norms and sanctions. With that, the RNS schema is highly sensitive to normative and sanctioning contexts.

2.3 Activity Types

According to the RNS schema, four types of activities are distinguished:

- Basic activities, that is, resource and event handling activities (*Type I*).
- Request activities, that is, requests for executing activities (*Type II*).
- Sanctioning activities, that is, activities that result in a punishment of behavior deviating from available obligations and interdictions, as well as activities that result in a rewarding of behavior going conform with permissions, obligations and interdictions (*Type III*).
- Change activities, that is, activities that result in changes of status statements being part of the status range of an activity of any type (*Type IV*). As a status statement consists of a norm specification and a sanction specification, change activities can be also characterized as activities that result *(i)* in changes of norms attached to an activity and/or *(ii)* in changes of sanctions associated with such norms.

Each of these four types of activities may be subject to (or "the target of") an activity of types II, III, and IV. This means, in particular, that the RNS schema allows to formulate "crossed and self-referential" constructs such as requests for requests, requests for sanction and norm changes, changes of norms attached to norm-changing activities (as well as requests for such changes), and changes of sanctions attached to sanction-changing activities (as well as requests for such changes). Examples of such constructs, which we call *activity reference constructs*, are provided below. In the following, the four activity types are described in detail.

Resource and Event Handling Activities. These activities are highly domain- and application-specific. Two types of resources are distinguished, namely, consumable ones (e.g., time, money, and any kind of raw material to be processed in a manufacturing process) and non-consumable ones (e.g., data, protocols, and communication support services such as blackboard platforms and translation systems). Examples of such activities are

provide(*CPU_time*), deliver(*material,quantity*), access(*database*),
run-protocol(*joint-planning*), kick-ball(*position*), acknowledge-receipt(*data*).

The RNS specification of this type of activities has the general form

```
ACT activity_id ( activity_variable_list )
   { STATUS RANGE status_statement_list }
```

where *activity_variable_list* is a list of variables specific to the activity *activity_id*. The first line of any activity specification, starting with the keyword ACT, is called an *activity header*, and the part enclosed in **{.}** is called an *activity body*. Here is an example of a specification of a basic activity. Assume there is a role with identifier USsupplier, and that one of its basic activities is specified as follows:

```
ACT deliver ( material,quantity )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
        <DEP EACH> : NORM <O> <quantity ≤ 100> + SANC <PU> <withdraw_role>
        <DEP AssemblyMg> : NORM <I> <material = steel> + SANC <PU> <pay_fine>
    }
```

The keyword EACH used as an instantiation of *role_id* (*agent_id*) indicates that *all* roles (agents) are concerned, and the keyword NO used as an instantiation of *condition* (of *sanction_type* and *sanction*) indicates that the norm is unconditioned (that there is no associated sanction). With that, in this example the IS statement says that an agent owning the role of which the deliver activity is part of is permitted to deliver. The first DS statement says that a request from each agent (no matter what role she owns within the role space under consideration) for executing this deliver activity induces the obligation to deliver, provided that the requested quantity is not above 100. Furthermore, the statement says that the requested agent must withdraw the role USsupplier (i.e., is not longer allowed to act as a USsupplier) in the case of violating such an induced obligation. The second DS statement says that the delivery of steel, if requested by an agent owning the role AssemblyMg ("Assembly Manager"), is forbidden; not acting in accordance with this interdiction is punished by some fine.

Execution Requests. These activities are specified as follows:

```
ACT REQUEST ( agent_id_list ; role_id_list ; [NOT] activity_id ( activity_variable_list ) )
    { STATUS RANGE status_statement_list
      NORMATIVE IMPACT norm_specification_list }
```

The activity header says that requests can be directed towards any agent who is referred to in *agent_id_list* and who owns at least one of the roles listed in *role_id_list*. The header also identifies the activity being subject to the request. The keyword NOT is optional and is to be used only in the case of interdiction (i.e., in the case of requests for not executing some activity). *norm_specification_list* specifies the normative impact of the request on the requested agent(s) through a list of norm specifications. As already introduced above, these specifications are of the form

```
NORM <norm_type> <condition>
```

Note that every norm specification included in a normative impact specification of a request activity, together with the identifier of the role of which the request activity is a part, unambiguously points to a single or (if there are multiple sanctions – rewards and punishments – associated with the induced norm) several DS statements.

As an illustrating example based on the delivery activity specified above, consider the following request activity specification being part of the role AssemblyMg:

```
ACT REQUEST ( EACH ; USsupplier, EUROsupplier ; NOT deliver ( material, quantity ) )
    { STATUS RANGE
        <IND> : NORM <P> < (material = steel) AND (rating(material) = poor)> +
                SANC <NO> <NO>
      NORMATIVE IMPACT
        NORM <I> <material = steel>
    }
```

The keyword **EACH** says that the request can be directed towards each agent owning the roles USsupplier or EUROsupplier. (If *role_id_list* were also instantiated with **EACH**, then this would mean that each agent – without any role restriction – can be requested to deliver.) Generally, the keyword **EACH** serves as a wildcard, and expressions including it are called *templates*.

A potential, legal occurrence or "call" of this request activity (which, for instance, could be part of interaction protocols) during run time is the following:

```
REQUEST ( Dr_Meyer, Mr_Black ; USsupplier ; NOT deliver ( steel, [0 . . . 500] ) )
```

(i.e., the USsuppliers Dr_Meyer and Mr_Black are requested to not accept steel delivery orders with an ordering volume less than or equal to 500 units)

As a variant of this example, consider the following specification (again assuming that the specified request activity is part of the AssemblyMg role):

```
ACT REQUEST ( EACH ; USsupplier, EUROsupplier ; NOT deliver ( material, quantity ) )
    { STATUS RANGE
        <IND> : NORM <P> < (material = steel) AND (rating(material) = poor)> +
                SANC <NO> <NO>
        <DEP MemBoardDirectors> : NORM <O> <NO> + SANC <PU> <reprimand>
      NORMATIVE IMPACT
        NORM <I> <material = steel>
    }
```

The status range of this variant includes a DS statement, meaning that this request activity becomes obligatory for an agent owing the role AssemblyMg if it is requested by an agent owning the role MemBoardDirectors ("Member of Board of Directors"). The specification of the corresponding request activity of the MemBoardDirectors role could look like this:

```
ACT REQUEST
    ( EACH ; AssemblyMg ;
      REQUEST ( EACH ; USsupplier, EUROsupplier ; NOT deliver ( material, quantity ) ) )
    { STATUS RANGE
        <IND> : NORM <O> <decided_by_board> + SANC <PU> <board_exclusion>
      NORMATIVE IMPACT
        NORM <O> <NO>
    }
```

With that, the RNS schema offers the possibility to formulate "requests for requests for requests for . . .", that is, *nested requests*.

Sanctioning Activities. Activities of this type are specified as follows:

```
ACT SANCTION ( agent_id_list ; role_id_list ; activity_id ; norm_spec )
    { STATUS RANGE status_statement_list
      SANCTIONING IMPACT sanction_specification_list }
```

where *norm_spec* is a norm specification and *sanction_specification_list* is a list of sanction specifications, that is, a list of specifications of the form

```
SANC <sanction_type> <sanction>
```

The sanctioning impact part specifies all sanctions that "become reality" through the execution of the sanctioning activity. Here is a simple example of a sanctioning activity, based on the "deliver" activity specification above:

```
ACT SANCTION ( EACH ; EACH ; deliver ; NORM <O> <quantity ≤ 100> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <RE> <earn_bonus>
        <DEP RoleSpaceMg> : NORM <I> <NO> + SANC <PU> <withdraw_role>
      SANCTIONING IMPACT
        SANC <PU> <withdraw_role>
    }
```

The two occurrences of EACH indicate that the sanctioning activity concerns each agent and each role of which the activity with identifier "deliver" is part of. The IS statement says that an agent owning a role of which this sanctioning activity is part of is unconditionally permitted (i.e., may) to execute this sanction, and that she earns some bonus in the case she does (i.e., actually makes use of her permission). The DS statement says that the sanctioning activity may become subject to an unconditioned interdiction, namely, as the result of an "interdiction request" by an agent owning the role with identity RoleSpaceMg ("Role Space Manager"); violating such an interdiction is punished through the withdrawal of role ownership.

An example of a real-time occurrence (instantiation) of this sanction specification is

```
ACT SANCTION ( Dr_Meyer ; USsupplier ; deliver ; NORM <O> <quantity ≤ 100> )
```

A further example of a sanction specification illustrating the expressiveness of RNS is the following. Under the assumption that each norm violation is punished by the withdrawal of role ownership, the "most general" specification of a sanction activity that can be constructed is

```
ACT SANCTION ( EACH ; EACH ; EACH ; EACH )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
      SANCTIONING IMPACT
        SANC <PU> <withdraw_role>
    }
```

saying that each agent owning a role of which this activity specification is part of is unconditionally permitted to sanction any norm violation through the withdrawal of role ownership.

Change Activities. Change activities affect the status range of activities. Three types of change activities are distinguished: DEL (delete), REP (replace), and ADD (add). The specification of these activities is as follows:

```
ACT ADD ( role_id_list ; activity_id_list ; status_statement )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        add status_statement ]
    }
ACT DEL ( role_id_list ; activity_id_list ; status_statement )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        delete status_statement ]
    }
ACT REP ( role_id_list ; activity_id_list ; status_statement_1 ; status_statement_2 )
    { STATUS RANGE status_statement_list
      [ STATUS IMPACT
        replace status_statement_1 by status_statement_2]
    }
```

The status impact parts are optional as they are of explanatory nature only. Here is an example of a specification of a change activity:

```
ACT REP
    ( USsupplier, EUROsupplier ; deliver ;
      <IND> : NORM <P> <NO> + SANC <NO> <NO> ;
      <IND> : NORM <O> <NO> + SANC <PU> <pay_fine> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
    }
```

An agent owning a role which includes this activity specification is permitted to replace, within each deliver activity being part of the roles USsupplier and EUROsupplier, the first status statement given in the activity header by the second one. (Note that the first IS statement in the header and the IS statement in the status range part are syntactically identical.)

Another example of change activity specification is the following:

```
ACT DEL
    ( USsupplier ; deliver ;
      <IND> : NORM <P> <NO> + SANC <NO> <NO> )
    { STATUS RANGE
        <IND> : NORM <P> <NO> + SANC <NO> <NO>
        <DEP USdirector> : NORM <I> <NO> + SANC <PU> <withdraw_role>
    }
```

This change activity concerns the deletion of the status statement

```
<IND> : NORM <P> <EACH> + SANC <EACH> <EACH>
```

attached to the deliver activity of the USsupplier role (as specified in the activity header). As can be seen from the status range of the delete activity specification, an agent owning the role USdirector may forbid this delete activity (i.e., is authorized to request to not execute it). Generally, RNS enables to formulate requests on change activities, and, reversely, it enables to formulate changes of status statements belonging to request activities.

Finally, here is an example of a specification of an add activity:

```
ACT ADD ( EACH ; EACH ;
            <DEP President> : NORM <I> <NO> + SANC <PU> <role_space_exclusion> )
     { STATUS RANGE
          <IND> : NORM <P> <NO> + SANC <NO> <NO>
     }
```

The owner of a role containing this activity specification is permitted to add the specified DS statement to every activity being part of any role (in the role space). Once added to an activity (more precisely, to the status range attached to an activity), an agent being in the role of President may unconditionally forbid this activity, where the consequence of violating this interdiction is the exclusion from the role space.

3 Discussion

A key feature of the RNS schema is its domain- and application independence. This means that the schema can be used to specify the kind and degree of autonomy which the agents should possess, no matter what specific application is considered and no matter in what domain this application runs. RNS is particularly suited for requirements elicitation and analysis, and thus its primary use lies in the early phase of agent-oriented systems development. More specifically, there are several potential advantages resulting from a use of RNS during requirements elicitation and analysis: it both enables and forces a developer to carefully reflect on the role of autonomy and autonomous behavior; it helps to avoid misunderstandings among analysts, stakeholders and programmers when communicating about agent autonomy; it facilitates the detection of potential faults caused by autonomous behavior; and, last but not least, it may serve as a basis for clarifying legal liability issues posed by agents acting autonomously on the behalf of human users.

Another key feature of RNS is that it is strongly expressive and enables a highly precise specification of agent autonomy. Expressiveness and precision derive from the following features:

- Through its concept of (positive and negative) sanctions RNS enables a developer to explicitly specify consequences of both norm-conforming and norm-deviating behavior. The importance of specifying these consequences results from the fact that autonomy, taken seriously, implies *autonomy against norms* [4] – an agent as an autonomous entity can not be guaranteed to act in accordance with all available norms under all circumstances.

- Through its concept of change activities RNS supports the explicit modeling and specification of potential *dynamic changes* in norms and sanctions and thus in behavioral autonomy.
- Through its concept of a status range RNS enables a developer to specify different normative impacts on the same activity. This makes it possible to cope with situations in which the normative status of an activity depends on the request context, that is, on who requested the activity under what condition. With that, RNS allows to explicitly capture *context sensitivity* of norms and thus of autonomy.
- RNS supports the specification of complex activities through various activity reference constructs. While some possible reference constructs (e.g., "a request for requesting a certain resource handling activity") may be only of marginal interest in an application at hand, others (e.g., "a request for sanctioning a norm violation") may be of particular importance.
- As it is based on the role concept, RNS does not imply constraints on the type and structure of the individual agents. Instead, it enables a developer to abstract from architectural aspects of agency. This is of particular importance in view of open applications.

Further appealing features of the RNS schema are the following: it is based on a relatively simple and intuitively clear notation and syntax; it is fully neutral w.r.t. autonomy (i.e., it is neither biased in favor of nor against autonomy and so supports a developer in specifying any autonomy level she considers as appropriate); and it is grounded in sociological role theory.

It is important to see that RNS operates on the agent and role level, although it also may be of use for norms-based interaction protocol specification; however, this is an open issue that remains to be investigated. A difference between RNS-type specification and protocol-type specification seems to be in the potential normative impact of requests: in the case of RNS, a request may induce an obligation; against that, in the case of (speech act-based) interaction protocols it is typically assumed that an obligation is not induced by a request per se (i.e., from the pure fact that something is requested), but by the – explicit – acceptance of a request.

There are several approaches which are closely related to RNS in that they also aim at a norms-based specification of autonomous behavior [6, 7, 9, 2, 5]. As elucidated below, what makes RNS distinct from all these approaches is the expressiveness and precision with which it allows to capture autonomy.

An approach which shows several interesting parallels to RNS is described in [6]. The focus there is on norm compliance and on the question what motiviations an agent might have to comply with norms. Like RNS, this approach is based on the view that agents as autonomous entities may decide to not act in accordance with norms; moreover, similar to RNS this approach considers the issue of positive and negative sanctions. The main difference is that this approach does make several strong and in some sense restrictive assumptions on the cognitive structure and processes within the individual agents (e.g., by treating sanctions as the agents' goals and by defining autonomy in terms of motivations hold by agents). Against that, RNS does not make restrictive assumptions on "things occuring within agents", but concentrates on the role level.

Another approach showing interesting parallels to RNS is presented in [7]. This approach focuses distributed systems management through policies. A policy in this approach is understood as a behavior-influencing information being located outside of the managers themselves, and is specified in terms of normative concepts (authorizations and obligations). Similar to RNS, this approach employs the role concept and supports a specification of context sensitivity. The main differences are that this approach does assume that agents always do behave norm-conforming (thus sanctioning is not considered), that complex activity specification is not supported, and that the specification of dynamic norm (and sanction) changes is not supported.

A logic-based approach related to RNS is described in [9]. This approach concentrates on collective agency and offers, similar to RNS, a normative system perspective. One important difference is that RNS, in contrast to this approach with its roots in deontic logic, does not rely on inter-definability of permissions and obligations (i.e., $P(x) =_{def} \neg O \neg x$). Another important difference is that this approach does neither consider the possibility of norm-deviating behavior nor the issue of dynamic norm change activities.

Other logic-based approaches related to RNS are described in [2] (dealing with norms-based coordination) and [5] (dealing with norms-based capturing of autonomous agents from a more general perspective). Like RNS, these approaches employ the concepts of permissions, obligations and interdictions and consider sanctions on norm-deviating behavior (though only negative sanctions). Unlike RNS, the approaches do not support the specification of dynamic changes in norms and sanctions and do not capture complex activity specification. Another difference is that these approaches are not role-based; instead, norms and sanctions are directly attached to agents and assumptions are made on agent-internal (cognitive) processes.

4 Conclusion

It should be clear that RNS in its current form leaves room for improvement. The two most critical deficiences of RNS we identify are the following. First, RNS does not support developers in explicitly specifying information and control relationships among roles such as generalization, aggregation, inheritance, peer, superior-subordinate, and so forth. Without support of such an explicit specification is it difficult (especially for large-scale applications) to obtain transparency of the overall system and its internal organizational structure. Second, RNS does not support developers in identifying and avoiding conflicts among norms (e.g., permission and interdiction of the same activity). Especially for large-scale applications such a support is extremely important as a means for avoiding poor system behavior resulting from normative conflicts. Both deficiences are of particular relevance w.r.t. a coherent and consistent system perspective and both require to extend RNS appropriately. What needs to be done in a first step thus is to define clear and useful conceptualizations of role-role relationships and normative conflicts. Encouraged by the above mentioned advantages of RNS we are currently concentrating on this first step. Moreover, we are also working on a software tool which supports RNS-based systems specification during requirements elicitation and analysis; a key service this tool is planned to offer is the automated detection of norm-based conflicts.

Acknowledgements

This work has been supported by Deutsche Forschungsgemeinschaft (DFG) under contract Br609/11-2. We would like to thank the reviewers for their valuable comments.

References

[1] R. Alterman. Rethinking autonomy. *Minds and Machines*, 10(1):15–30, 2000. 83

[2] M. Barbuceanu, T. Gray, and S. Mankovski. The role of obligations in multiagent coordination. *Journal of Applied Artificial Intelligence*, 13(2/3):11–38, 1999. 93, 94

[3] B. J. Biddle and E. J. Thomas, editors. *Role theory: Concepts and research*. John Wiley & Sons, Inc., New York, London, Sydney, 1966. 84

[4] R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm acceptance. In J. P. Müller, M. P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 99–112. Springer-Verlag, 1999. 83, 92

[5] F. Dignum. Autonomous agents with norms. *Artificial Inteligence and Law*, 7:69–79, 1999. 83, 93, 94

[6] F. Lopez y Lopez, M. Luck, and M. d'Inverno. Constraining autonomy through norms. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS'2002)*, 2002. 93

[7] E. Lupu and M. Sloman. Towards a role based framework for distributed systems management. *Journal of Network and Systems Management*, 5(1):5–30, 1997. 93, 94

[8] D. Musliner and B. Pell (Cochairs). Agents with adjustable autonomy. Papers from the AAAI spring symposium. Technical Report SS-99-06, AAAI Press, Menlo Park, CA, 1999. 83

[9] O. Pacheco and J. Carmo. A role based model for the normative specification of organized collective agency and agents interaction. *Journal of Autonomous Agents and Multi-Agent Systems*, 2002. to appear. 93, 94

Activity Theory as a Framework for MAS Coordination

Alessandro Ricci[1], Andrea Omicini[1], and Enrico Denti[2]

[1] DEIS, Università di Bologna
Via Rasi e Spinelli 176, 47023 Cesena, FC, Italy
{aricci|aomicini}@deis.unibo.it
[2] DEIS, Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
edenti@deis.unibo.it

Abstract. Approaches to the coordination of multiagent systems (MAS) have been recently classified as *subjective* – typically coming from the distributed artificial intelligence (DAI) –, and *objective* – coming from the community of Coordination Models and Languages. Subjective and objective approaches have a very different impact on the engineering of social aspects of MAS, in particular with respect to the ability of specifying and enacting social laws to achieve global coherent behaviours. In this work, we provide a conceptual framework – influenced by the research on *Activity Theory* – where both subjective and objective coordination play an essential role, each providing effective means for the same coordination/cooperative problems at different abstraction and operational levels: *co-construction*/*co-operation* level for subjective coordination, and *co-ordination* level for objective coordination. In particular, the work shows the benefits of supporting dynamic transitions between such levels, alternating co-operation stages – in which agents reason about coordination and cooperatively forge coordination *artifacts* (laws, constraints, norms) – and co-ordination stages – where the artifacts, embodied in proper coordination media, are exploited, so as to enact automated, consistent and prescriptive coordination.

1 Objective vs. Subjective Coordination in MAS

Interaction and coordination are interdisciplinary issues, and therefore it is not surprising that their study and development in multiagent systems (MAS) is supported by approaches coming from heterogeneous contexts (refer to [22, 13, 38, 28, 29] for comprehensive surveys).

The major contribution to coordination in MAS comes from the distributed artificial intelligence (DAI) field. Generally, DAI approaches explicitly deal with *subjective coordination* [34], considering coordination as "the process by which an agent reasons about its local action and the (anticipated) actions of others to try and ensure the community acts in a coherent manner" [13]. Consequently, the coordination of the overall system is determined by both the mental attitudes

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 96–110, 2003.

(beliefs, goals) and the role of individuals, and by the subjective perception of the inter-dependencies among the members of a society. Well-known examples of subjective coordination techniques in MAS are *multi-agent planning* [10], where agents build plans and commit to behave in accordance with them, and *negotiation* [4, 22], where agents communicate so as to reach a mutually accepted agreement on some matter. Typically, subjective coordination is exploited in MAS composed by intelligent agents, provided with an high-level agent communication language (ACL), such as KQML or FIPA, whose formal semantics can become the means to realise flexible coordination [3].

Other substantial contributions to MAS coordination are rooted in concurrent (parallel and distributed) systems. Starting from the need to explicitly separate coordination from computation issues in system design and development [12], several *coordination models and languages* have been developed [30], and applied to MAS [29]. Generally, these approaches explicitly deal with *objective coordination* [34], promoting the separation between the individual perception of coordination and the global coordination issues, enabling the modelling and shaping of the interaction space independently of the interacting entities. This kind of coordination is called *objective* because it prescinds from the subjective view of the coordinated agents. It is also called *uncoupled* [38], since coordination is no longer coupled with the computational issues of the coordinated entities. According to [34], objective coordination is mainly concerned with inter-agent issues such as *(i)* the description of how the MAS environment is organised, and *(ii)* the management of interactions between both agents and their environment, and agent themselves. Objective coordination models are heavily based of the concept of *mediated interaction*: agents are provided with specific abstractions that enable their actions (typically communications), and mediate the emerging interactions, caused by the dependencies inside the agent ensemble. In the classification generally adopted by the coordination community [6], this abstraction role is assumed by the *coordination medium*, which rules agent interactions by applying *coordination laws*, representing social laws and system constraints. *Tuple centres* [25] and the related TuCSoN coordination infrastructure [26] are an example of a coordination approach promoting objective coordination.

The variety and complexity of the interaction scenarios that characterise multiagent systems and agent societies require approaches with the properties of both subjective and objective coordination. For instance, distributed workflow management systems (WfMS) and office automation environments require the automatism and prescriptiveness that are easily provided by objective approaches; instead, unstructured environments (from the coordination point of view) and market-oriented competitive scenarios mostly rely on the reasoning and interaction capabilities of individual agents. Moreover, relevant coordination scenarios, such as business process coordination and computer supported cooperative work (CSCW), require models balancing subjectivity and objectivity, and mediating between application-centric and human-centric tasks, unstructured process and highly structure process structure [8]. From both a scientific and an engineering viewpoint, it is then important to provide a conceptual framework exploiting both the subjective and the objective coordination approaches, where to ground models and infrastructures for MAS coordination.

The interdisciplinary nature of interaction and coordination makes it possible to find relevant contributions in disciplines and theories outside computer science: social theories about organisation and coordination in human societies can be very effective, given the complexity of the interaction space involved in some MAS scenarios. An example is the theory of coordination developed by the Centre of Coordination Study (CCS) at MIT: from the analysis of heterogeneous contexts, from economy to computer science, coordination emerges as the process of *managing dependencies among activities* [18]. Accordingly, engineering coordination inside systems means first identifying the dependencies among the individual parts, then defining strategies to manage the identified dependencies (or interactions, when dependencies are among agent actions), and finally enacting the strategies. The concept of *dependency* is a very important element for both subjective and objective approaches: in the first case of subjective approaches, dependency appears as the basic relationship on which the construction of the whole agent social life is founded [5, 7], while objective coordination approaches are explicitly focused on the management of dependencies/interactions.

The coordination theory approach is clearly *bottom-up* (regulatory): dependencies are the starting point, then coordination activities are designed and developed on top of dependency analysis. However, the social issues that we aim at engineering in MAS also account for a *top-down* (constructive) approach to coordination: the starting point here are the social goals and the properties that must be provided by the aggregation of agents as a system/society, then dependencies among the individual parts are determined/induced accordingly.

While the theory of coordination provides a good conceptual background for the bottom-up approach, we found *Activity Theory* [39] very effective to frame the role of objective and subjective coordination (and their relationship) in the top-down case. Initially developed to study the dynamics of collective human work activity, Activity Theory has been recently introduced in computer science disciplines, such as CSCW and HCI (Human Computer Interaction). Both activity and dependency theories refer (either directly or indirectly) to the notion of mediated interaction: interaction media are studied, catalogued and used to manage dependencies in the theory of coordination, and media are forged as coordination *artifacts* in Activity Theory. The need to define abstractions and "social structures" mediating agent interaction is clearly visible also in the evolution path of some subjective coordination approaches in MAS: the notion of *social agency* [36] and *social laws* [35], the STEAM teamwork module in the TeamCore coordination architecture [37], and stigmergy coordination [31] are notable examples. In the last case, in particular, agent interactions are mediated and ruled by the environment, with its own laws and constraints, and coordination appears as an emergent phenomenon. Elements related to the objective approach are clearly distinguishable also in some works included in this volume: for instance, the *external observer* perspective of computational societies [14], using the *Society Visualiser* to focus inter-agent aspects (macro level) despite of intra-agent ones (micro level); or the *role space* – defined in the RNS schema specification [20] – where social norms are externalised, ruling objectively the inter-agent behaviour of the agent society/organisation.

Given these premises, in this seminal work we provide a conceptual framework derived from Activity Theory where both subjective and objective approaches are exploited in the same coordination context, but at different conceptual and operational levels; the framework gives insights into the dynamics between the approaches/levels, showing the fundamental role of mediated interaction (exemplified by the coordination medium/artifact abstractions) for that purpose. Three hierarchical levels for analysing every social activity in MAS are identified: *co-construction*, *co-operation* and *co-ordination*. Accordingly, we show how subjective and objective approaches are both fundamental to support such levels: in particular subjective approaches for co-construction and co-operation, and objective ones for co-ordination. In this way, both high level cooperation protocols among intelligent agents – typically found in subjective approaches – and scripted coordination driven by laws embedded in coordination media – as typically found in objective coordination – turn out to be necessary to build MAS with autonomous agents behaving efficiently and coherently in a social/systemic context. Moreover, the dynamics between co-operation and co-ordination is discussed, providing the notion of coordination reflection and reification.

The remaining part of the work is organised as follows: Section 2 provides the basic elements of Activity Theory useful for this work. Section 3 comes back to coordination in MAS, and provides a conceptual framework – derived from previous section – in which co-ordination/co-operation activities are analysed, and the relationship with subjective and objective coordination discussed. Finally, Section 4 provides conclusions and future works.

2 Activity Theory

Activity Theory (AT) is a social psychological theory about the developmental transformation and dynamics in collective human work activity [39, 17, 2]. Recently, AT has been introduced in some fields of computer science – in particular in CSCW [16] and computer system design [19]. AT focuses on *human activities*, distinguished by their respective (physical and ideal) *objects*, that give them their specific directions, i.e. the *objectives* of the activities. Cooperation is understood as a *collaborative activity*, with one objective, but distributed onto several actors, each performing *actions* accordingly to the shared objective. Explicit norms and rules regulate the relationships among individual participants' work. Central to AT – as in the case of objective coordination – is the notion of mediated interaction: human activity is found to be always mediated by *coordination artifacts*, both physical and psychological, such as operating procedures, heuristics, scripts, individual and collective experiences, and languages. In this context, (scripted) *roles* can be understood as actors' expected behaviour with respect to *scripts*, that are the coordination artifacts embodying the shared objectives.

2.1 Co-operation and Co-ordination in AT

AT identifies three hierarchical levels defining the structure of collaborative activities: *co-ordinated*, *co-operative*, *co-constructive* [2, 11]:

- *co-ordinated* aspect of work captures the normal and routine flow of interaction. Participants follow their scripted roles, each focusing on the successful performance of their actions, implicitly or explicitly assigned to them; they share and act upon a common object, but their individual actions are only externally related to each other. Scripts coordinating participants' actions are not questioned or discussed, neither known/understood in all their complexity: in this stage actors act as "wheels in the organisational machinery" [16], and co-ordination ensures that an activity is working in harmony with surrounding activities.
- *co-operative* aspect of work concerns the mode of interactions in which actors focus on a common object and thus share the objective of the activity; unlike previous case, actors do not have actions or roles explicitly assigned to them: with regard to the common object, each actor has to balance his/her own actions with other agent actions, possibly influencing them to achieve the common task. So, in this case the object of the activity is stable and agreed upon: however the means for realising the activity is not yet defined.
- *co-constructive* aspect of work concerns interactions in which actors focus on re-conceptualising their own organisation and interaction in relation to their shared objects. Neither the object of work, nor the scripts are stable, and must be collectively constructed, i.e. *co*-constructed.

In the analysis of collaborative activities, AT emphasises that an activity cannot be said to exist on one level alone: co-ordination, co-operation, and co-construction are *analytical* distinctions of the same collaborative activity, and concur in different times and modes to its development.

2.2 Dynamic Transformation between Collaborative Levels

The notion of dynamic transformation between the hierarchical levels is central to AT. Transformations are strictly related to the stability of the means of work and of the object of work (see Fig. 1, taken from [2]): upward transformations are caused by reflection on the means for doing the work or on the object of the work itself, while downward transformations are caused by resolving conflicts and problems, and re-embodying the solution in the lower levels.

In the context of this article, *reflection* on the means of work – going from co-ordination to co-operation, and *routinisation* – going from co-operation to co-ordination – are the most important transitions. The former happens when the coordinated flow of work, relying on stable means of work such as scripts, rules,

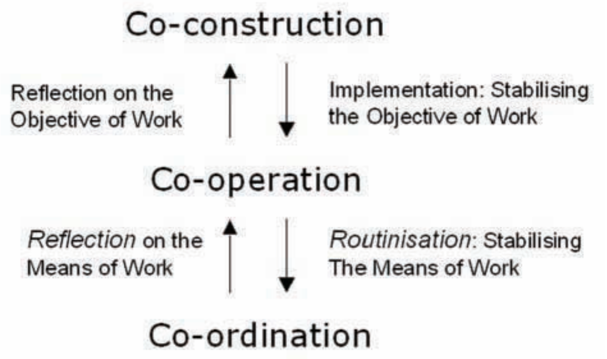

Co-construction
Reflection on the Objective of Work
Implementation: Stabilising the Objective of Work
Co-operation
Reflection on the Means of Work
Routinisation: Stabilising The Means of Work
Co-ordination

Fig. 1. Dynamics of the cooperative work as reported in [2]

coordination artifacts in general, needs to be cooperatively re-established according to the object of work; the reasons can be either coordination breakdown, or a deliberate re-conceptualisation of the way the work is achieved currently. The latter works in the opposite directions, by re-establishing co-ordinated work where the means of collaboration are stabilised, and (mediating) artifacts are provided to be exploited by participants in the co-ordination stage.

The other transitions account for reflecting on the object of work – when the objective become unstable within the collaborating ensemble and transformation to the co-constructive level of collaboration is necessary –, and for implementation – when conflicts on a common objective are resolved, and communication/negotiation ensuring its commitment can begin.

3 Back to MAS: Reconciling Objective and Subjective Approaches

The conceptual framework provided by the AT is effective here to understand the (distinct) roles of objective and subjective coordination in MAS, and the relationships that occur between them. We reconsider the hierarchical structure of collaborative activities provided by the AT, and contextualise it according to the theory of coordination. Accordingly, in context of MAS coordination three main stages can be identified (see Fig. 2):

- **co-construction** – about the understanding and reasoning on what dependencies and interactions have to be faced and managed;
- **co-operation** – about planning what actions must be taken to manage the dependencies and interactions that have been identified in the previous (co-construction) stage;
- **co-ordination** – about enforcing/automating the activities to manage the interactions, planned in the co-operation stage.

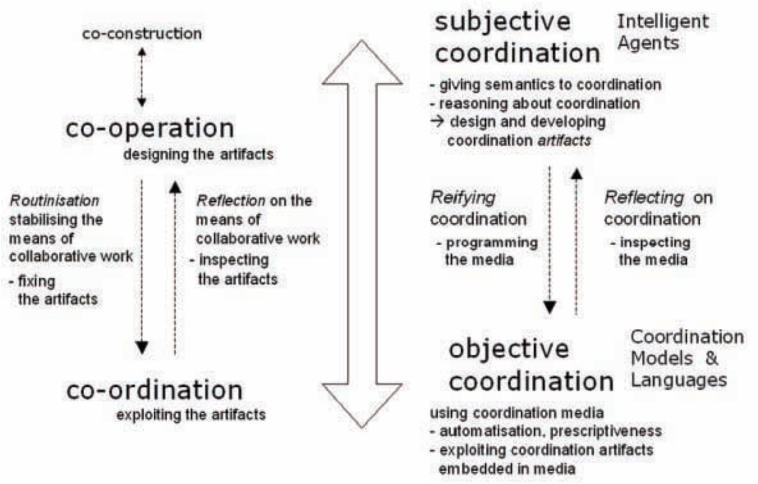

co-construction
co-operation
designing the artifacts
Routinisation
stabilising the means of collaborative work
- fixing the artifacts
Reflection on the means of collaborative work
- inspecting the artifacts
co-ordination
exploiting the artifacts
subjective coordination
Intelligent Agents
- giving semantics to coordination
- reasoning about coordination
→ design and developing coordination artifacts
Reifying coordination
- programming the media
Reflecting on coordination
- inspecting the media
objective coordination
Coordination Models & Languages
using coordination media
- automatisation, prescriptiveness
- exploiting coordination artifacts embedded in media

Fig. 2. Dynamics between objective and subjective coordination

Given these three levels, subjective approaches can be considered fundamental for the co-operation stage. Here (intelligent) agents are necessary to reason about what kind of coordination is required, what kind of coordination laws must be developed to manage interactions identified in co-construction stage. The smartness of the agents is useful to build cooperatively – by means of negotiation and high level (semantics driven) interaction protocols – effective artifacts (i.e. coordination laws defining coordination media behaviour) to be used in the co-ordination stage.

Instead, objective coordination is fundamental for the co-ordination stage, where the coordination laws and organisational rules must be enacted in the most automated, fluid, optimised manner. The coordination medium abstraction (and coordination laws defining its behaviour) represents and embodies effectively the AT concept of artifact (and related mediating tools), embedding and enacting in the co-ordination stage the social laws and interaction constraints established in the co-operation stage. The parallel between AT artifacts and coordination media help to understand the role of the media inside MAS: as the artifacts, coordination media first are used to *enable* the interactions among the agents, and then to *mediate* them in the most automated manner. As the artifacts, media become *the* place where the (partial) knowledge about the global behaviour of the MAS is traced and can be further inspected. As the artifacts, media become the source of the *social intelligence* that characterises concretely the systemic/synergistic vision (as opposite of compositional) of MAS [6]; in this context, coordination laws become the macro-levers that can be used to tune and adapt dynamically such a collective intelligence.

The explicit identification of a co-ordination stage distinct from the co-operation stage, with objective coordination models exploited in the first case and subjective models in the latter, is useful to understand the relationships between agent autonomy and social order, between the autonomy of single behaviours and the achievement of a globally coherent behaviour. Agent autonomy is preserved both in the co-operation stage – essential to build the social laws according to the aims and capability of the single agents –, and in the co-ordination stage – where agents are not constrained to know/understand all the coordination machinery (embedded in media) in order to participate to coordination. In particular, in the co-ordination stage agents can focus on the actions related to their specific goals, and delegate implicitly to the medium itself (by means of its role of social mediator) the enactment of the laws that characterise the global coordination activities. In this way, also agents without high level coordination skills can exploit the coordination service; indeed, this is a very important aspect in the engineering of (open) MAS: the support for effective coordination among agents with heterogeneous computational models, not necessarily intelligent or BDI, not necessarily speaking with an high level ACL. The same thing happens in AT collaborative scenarios: often humans exploit coordination artifacts without knowing their whole behaviour, focusing only on the actions that are expected from them by means of their role.

An other important coordination issue taken into account within this conceptual framework is the performance / optimisation of coordination activities. Suitably designed coordination media make it possible to minimise the interactions involved in the co-ordination stage: medium laws automate the coordination flow, minimising the need of negotiation among coordinated agents – which typically characterises the co-operation stage. A good cooperation is then fundamental to produce effective laws that can affect the performance of the global behaviour of the MAS; of course, the impact depends also on the expressive power of the coordination medium, which constrains the possible social laws that can be specified and enforced. Also in this case, similarities with the human case are evident: the effectiveness of the routinised-work in the co-ordination stage depends on the characteristics of the artifacts used to mediate actors' interactions.

This coordination framework is strictly related to MAS planning/scheduling and execution approaches, with two important points that distinguish the former from the latter at the foundation level: *(i)* the main focus is not on the individual agents and their computational/planning behaviour, but on the agent system and interactions; *(ii)* the coordination rules planned and scheduled in the co-operation stage do not "melt" into the agent computational behaviour in the co-ordination stage, but have a conceptual and physical body in the coordination media, outside agents, used by agents. It is worth to remark that Activity Theory has been exploited as a conceptual basis for *situated planning* [1], providing the connection between plans and the contextual condition for realising the plans in actual work.

3.1 Coordination Media Are Not Agents

Generally, in the context of MAS, agents are the only abstractions used for the system engineering – especially at the development and deployment stage. In particular, mediator services are provided by agents, too: *middle-agents* are a well-known example, providing coordinating activities (such as matchmaking, brokering) among agent providers and requesters (information, goods, or expertise), locating and connecting agents in open environments [15]. Accordingly, middle-agents can be understood as a suitable way to embody the AT artifacts at the co-ordination stage, and so acting as coordination media mediating agent interactions. However, as AT clearly distinguishes the ontological properties of the artifacts – and related mediating tools – and of the actors designing/developing (co-operation stage) and exploiting (co-ordination stage) the artifacts, the same approach can be adopted here for agents and coordination media. Among the main properties expected from a coordination medium there are:

- *inspectability* – the behaviour of a coordination medium should be inspectable, both from human and artificial agents. Moreover, the inspected coordination specification should be described in a declarative way, possibly with a formally defined semantics, to make the comprehension of coordination activities semantics for intelligent agents easier;
- *efficiency/specificity* – a coordination medium should be specialised in the management of interactions, in order to maximise performance in applying the coordination rules; moreover, a medium should be specialised to support the concurrent actions (communications) of multiple agents, possibly providing security, reliability and fault tolerance capabilities;
- *predictability* – the behaviour of a coordination medium should exactly reflect the coordination laws upon which it has been forged (autonomous, unpredictable behaviour is typically not desired); a formal semantics should be defined for the coordination model, in order to precisely define the effect of the coordination laws on the state of the medium and, more generally, on the agent interaction space;
- *flexibility* – a coordination medium should allow its behaviour to be forged dynamically.

Agents are generally not meant to provide such properties, since they are supposed to be autonomous, pro-active, situated entities, and typically speak by means of a general-purpose / high-level communication language [40].

These characteristics remark the conceptual and physical difference between the agent and the medium abstractions. In the model, for instance, coordination media are represented by programmable tuple spaces (or reactive logic-based associative blackboards, from the AI perspective) called tuple centres; agent interaction is enacted through the exchange of information via these spaces. The coordination laws that define the behaviour of tuple centres are expressed in the ReSpecT logic-based language [24, 25]: therefore, coordination artifacts are forged by suitably programming tuple centres, "engraving" the coordination media according to the need. As an example, the chunk of ReSpecT code in Fig. 3

```
reaction(out(task_end(first_task)),(
    in_r(task_end(first_task)),in_r(task_end(second_task)),out_r(new_task_to_do(third_task)))).
reaction(out(task_end(second_task)),(
    in_r(task_end(first_task)),in_r(task_end(second_task)),out_r(new_task_to_do(third_task)))).
```

Fig. 3. The coordination law enforced by the above ReSpecT code induces the execution of the task `third_task` after the completion of tasks `first_task` and `second_task`. An agent takes in charge a task execution by retrieving the tuple `new_task_to_do(`*`TaskId`*`)` from the tuple centre (by means of an `in` operation) and inserts the tuple `task_end(`*`TaskId`*`)` (by means of an `out` operation) when the task is done. So, according to the above code and the semantics of ReSpecT reactions [24], when an agent manifests the end of either `first_task` or `second_task`, the medium reacts and produces the tuple required for the execution of `third_task` *iff* both the tuples `task_end(first_task)` and `task_end(second_task)` are currently present in the tuple centre – that is, *iff* both tasks have been successfully brought to end.

enforces a simple coordination policy synchronising three activities possibly executed by three different agents.

In particular, ReSpecT coordination media provide the properties remarked above. As a general-purpose language for coordination, ReSpecT features a reaction-based computational model, and primitives specialised in the management of interactions / communications. Such a specialisation impacts on both expressiveness – since higher-level coordination pattern/policy can be easily described by composing the primitive elements – and efficiency – since the primitive elements that affect the coordination performance are clearly identifiable and tunable. ReSpecT formal semantics [24] makes it possible to exactly predict the behaviour of the coordination medium given a ReSpecT specification and the current tuple centre state. Also, inspectability of the coordination laws is made possible by their explicit representations in terms of ReSpecT tuples, and their embodiment within tuple centres, where they can be accessed by both agents – through either meta-level coordination primitives or high-level abstractions like *agent coordination contexts* [23] – and engineers – through specific deployment tools like the Inspector [9]. Finally, the behaviour of a tuple centre can be modified dynamically, by changing at run-time its ReSpecT specification, thus providing the required flexibility.

3.2 Reflecting and Reifying Coordination

The notion of dynamic transformation between levels in collaborative activities is central to AT. Accordingly, central to MAS coordination is the support for dynamic transformation from co-operation – that is the subjective coordination level – to co-ordination – that is the objective coordination level –, and vice versa. In particular, this is important in the context of *open* systems, where the environment is subject to change, and collective goals, norms, organisational rules must be adapted accordingly. This dynamism is captured by two basic transitions, the *reflection* and the *reification* of coordination, which must be supported dynamically, anytime required, during system execution. These transitions are strictly related to the transformations seen in the AT, and account for (see Fig. 2):

- **reification** – in this transition, coordination laws that have been designed and developed in the co-operation stage are reified in coordination media: intelligent agents forge / program the coordination media behaviour in order to reflect the social rules established in the co-operation stage, and to be used as artifacts in the co-ordination stage. It is worth noting that coordination media are meant to embed not only the rules promoting cooperation among agents, but also the laws ruling interactions, useful to represent also norms and environment constraints, mediating possible agent competitive (not cooperative) behaviour.
- **reflection** – in this transition, the behaviour of the coordination media deployed in co-ordination stage is inspected. Agents can retrieve and understand the coordination laws underlying medium behaviour, in order to either evolve them according to changes in coordination policies or in environmental conditions, or learn how to exploit efficiently the artifacts.

By making it possible to balance dynamically subjectiveness and objectiveness, providing the tools to establish at runtime the distribution of the coordination burden between media and agents, objective coordination models endorse a relevant engineering impact. As a useful picture, we can imagine a "coordination engineering segment" whose extreme points are on the one side all coordination intelligence upon agents, and on the other side all coordination burden upon media. The main issue here is to provide the expressive tools to locate, and dynamically move, the coordination engineering of the system on any point of the segment, thus determining a different morphology of coordination artifacts, and providing a smooth, by-need transition from subjective to objective orientation (see Fig. 4). The point position depends on both the considered coordination scenario and the dynamics inside that scenario: the more automation/prescriptiveness is required and the social rules are well-defined (such as for workflow systems), the more the coordination media are charged; the more in the coordination context it is not possible (or feasible) to clearly identify collective rules/constraints, the more the individual agents are charged of the coordination burden. In order to support these capabilities, coordination models and technologies must provide the means (languages and tools) for coordination reflection (from objective to subjective transition), to inspect the coordination laws defining media behaviour, and coordination reification (from subjective to objective transition), programming the behaviour of the coordination media.

The dynamic support for balancing task automation with cooperation flexibly is among the most important requirements for state-of-the-art systems for workflow management, supply chain management, and CSCW [8, 21]. The ability to change the "engineering point" of coordination dynamically is specially required in open systems, where the environment can unpredictably change, the goals of the system can evolve, and coordination laws can be improved as a result of agent interaction. In the case of TuCSoN, this is achieved by means of

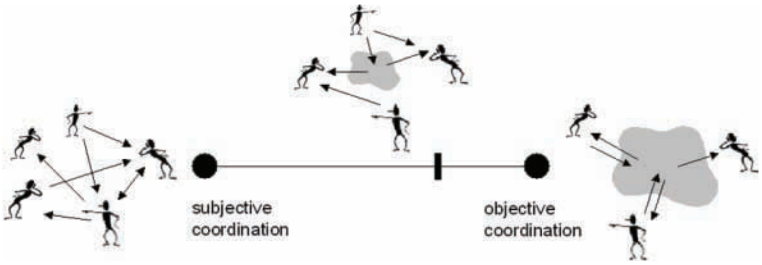


Fig. 4. The coordination engineering segment: all coordination in agents (subjective, left extreme), all coordination in media (objective, right extreme)

the ReSpecT tuple centre model, and the tools provided by the infrastructure. The coordination laws that define the behaviour of the coordination media (tuple centres) expressed in the ReSpecT language can be inspected and changed dynamically by human and artificial agents by means of specific tools. We are verifying the effectiveness of this approach in scenarios such as pervasive computing and inter-organisational workflow management systems [33]. In the last context, for instance, tuple centres have been used to embody the role of workflow engines, and workflow rules have been described as ReSpecT coordination laws. Each workflow engine (tuple centre) acts as a coordination artifact providing fluid coordination of the individual tasks executed autonomously by human and artificial agents. So, *(i)* workflow rules are inspectable by inspecting the coordination laws embedded in tuple centres (reflection stage); *(ii)* workflow rules are modifiable at runtime – as a consequence of exceptions, or changes in the business environment – by changing the coordination laws within tuple centres (reification stage).

4 Conclusions

Activity Theory provides a general framework for MAS coordination, which clarifies the distinct roles of subjective and objective approaches and their mutual relationship. By stressing the role of coordination artifacts, AT points out that models for MAS coordination, to be effective, should support both the subjective and objective viewpoints, and explicitly enable the dynamic transitions between the two – allowing coordination laws cooperatively established by agents to be fixed as coordination artifacts in proper media, and, conversely, coordination media to be inspected and their behaviour understood by agents.

Coordination infrastructures and tools are necessary to support the models at runtime, providing coordination as a service: this accounts for *(i)* providing human and artificial agents with the means for dynamically understanding what coordination media/artifacts are available and how they can be exploited (protocols, permissions, etc.); *(ii)* providing agents with tools to dynamically create, access and exploit the coordination media/artifacts [9].

Explicitly accounting for subjective and objective coordination also gives the opportunity to flexibly balance the intelligence/burden of coordination between the agents and the coordination services provided by the infrastructure. This can be done according to the level of automation and prescriptiveness needed, according to the skills of the agents involved in the coordination, and to the expressiveness of the media embodying/acting-as the artifacts. In particular, reflection and reification transitions allow the balance to be tuned and adapted dynamically, according to the changes required by the application scenarios.

References

[1] J. Bardram. Plans as situated action: An activity theory approach to workflow systems. In *Proceedings of the ECSCW 97 Conference*, Sept. 1997. 103

[2] J. Bardram. Designing for the dynamics of cooperative work activities. In *Proc. of the 1998 ACA4 Conference on Computer Supported Cooperative Work*, 1998. 99, 100, 101

[3] F. Bergenti and A. Ricci. Three approaches in coordination of MAS. In *Proc. of the 2002 ACM Symposium on Applied Computing (SAC'02)*. ACM, 2002. Track on Coordination Models, Languages and Applications. 97

[4] S. Bussmann and J. Mueller. A negotiation framework for co-operating agents. In D. S., editor, *Proocedings of CKBS-SIG*, pages 1–17. Dake Center, University of Keele, 1992. 97

[5] C. Castelfranchi, M. Miceli, and A. Cesta. Dependence relations among autonomous agents. In Y. Demazeau and E. Werner, editors, *Decentralized AI*, pages 215–231. Elsevier, 1992. 98

[6] P. Ciancarini, A. Omicini, and F. Zambonelli. Multiagent system engineering: the coordination viewpoint. In N. R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI — Agent Theories, Architectures, and Languages*, volume 1767 of *LNAI*, pages 250–259. Springer-Verlag, Feb. 2000. 97, 102

[7] R. Conte and C. Castelfranchi. *Cognitive and Social Action*. University College London, 1995. 98

[8] U. Dayal, M. Hsu, and L. Rivka. Business process coordination: State of the art, trends and open issues. In *Proceedings of 27th VLDB Conference*, 2001. 97, 106

[9] E. Denti, A. Omicini, and A. Ricci. Coordination tools for the development of agent-based systems. *Applied Artificial Intelligence*, 16(9), Oct. 2002. 105, 107

[10] E. Durfee. Planning in distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*. Wiley, 1996. 97

[11] Y. Engeström, K. Brown, L. Christopher, and J. Gregory. Coordinatio, cooperation, and communication in the courts. In Y. Engeström and O. Vasquez, editors, *Mind, Culture, and Activity*. Cambridge University Press, 1997. 100

[12] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, Feb. 1992. 97

[13] N. Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. Wiley, 1996. 96

[14] L. Kamara, A. Artikis, B. Neville, and J. Pitt. Simulating computational societies. In Petta et al. [32]. 98

[15] M. Klusch and K. Sycara. Brokering and matchmaking for coordination of agent societies: A survey. In Omicini et al. [27], chapter 8, pages 197–224. 104

[16] K. Kuutti. The concept of activity as a basic unit of analysis for CSCW research. In *Proceedings of the Second European Conference on CSCW*, pages 249–264. Kluwer Academic Publisher, 1991. 99, 100

[17] A. Leontjev. *Activity, Consciousness, and Personality*. Prentice Hall, 1978. 99

[18] T. Malone and K. Crowstone. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994. 98

[19] B. A. Nardi. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, 1996. 99

[20] M. Nikles, R. Michael, and G. Weiss. A schema for specifying computational autonomy. In Petta et al. [32]. 99

[21] G. Nutt. The evolution toward flexible workflow systems. *Distributed Systems Engineering*, 3(4):276–294, Dec. 1996. 106

[22] H. Nwana, L. Lee, and N. R. 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. 96, 97

[23] A. Omicini. Towards a notion of agent coordination context. In D. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, pages 187–200. CRC Press, 2002. 105

[24] A. Omicini and E. Denti. Formal ReSpecT. In A. Dovier, M. C. Meo, and A. Omicini, editors, *Declarative Programming – Selected Papers from AGP'00*, volume 48 of *Electronic Notes in Theoretical Computer Science*, pages 179–196. Elsevier Science B. V., 2001. 104, 105

[25] A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, Nov. 2001. 97, 104

[26] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999. Special Issue: Coordination Mechanisms for Web Agents. 97

[27] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, Mar. 2001. 109

[28] S. Ossowski. *Co-ordination in Artificial Agent Societies: social structures and its implications for autonomous problem solving agents*, volume 1535 of *LNAI*. Springer-Verlag, 1998. 96

[29] G. A. Papadopoulos. Models and technologies for the coordination of Internet agents: A survey. In Omicini et al. [27], chapter 2, pages 25–56. 96, 97

[30] G. A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers*, 46:329–400, 1998. 97

[31] V. D. Parunak. 'Go To The Ant': Engineering principles from natural agent systems. *Annals of Operations Research*, 75:69–101, 1997. 98

[32] P. Petta, R. Tolksdorf, and F. Zambonelli, editors. *Engineering Societies in the Agents World III*, LNAI. Springer-Verlag, Dec. 2002. 108, 109

[33] A. Ricci, A. Omicini, and E. Denti. Virtual enterprises and workflow management as agent coordination issues. *International Journal of Cooperative Information Systems*, 11(3/4):355–380, Sept./Dec. 2002. Cooperative Information Agents: Best Papers of CIA 2001. 107

[34] M. Schumacher. *Objective Coordination in Multi-Agent System Engineering – Design and Implementation*, volume 2039 of *LNAI*. Springer-Verlag, Apr. 2001. 96, 97

[35] J. Shoham and M. Tennenholtz. Social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73, 1995. 98

[36] M. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12), 1998. 98

[37] M. Tambe and D. V. Pynadath. Towards heterogeneous agent teams. In M. Luck, V. Mrik, O. Stepankova, and R. Trappl, editors, *Multi-Agent Systems and Applications*, volume 2086 of *LNAI*, pages 1–16. Springer-Verlag, 2001. 98

[38] R. Tolksdorf. Models of coordination. In A. Omicini, R. Tolksdorf, and F. Zambonelli, editors, *Engineering Societies in the Agents World*, volume 1972 of *LNAI*, pages 78–92. Springer-Verlag, Dec. 2000. 96, 97

[39] L. S. Vygotskij. *Mind and Society*. Harvard University Press, 1978. 98, 99

[40] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995. 104

An Operational Framework for the Semantics of Agent Communication Languages

Giovanni Rimassa[1] and Mirko Viroli[2]

[1] AOT Lab - Dipartimento di Ingegneria dell'Informazione,
Parco Area delle Scienze 181/A, 43100 Parma, Italy
rimassa@ce.unipr.it

[2] DEIS, Università degli Studi di Bologna,
via Rasi e Spinelli 176, 47023 Cesena, Italy
mviroli@deis.unibo.it

Abstract. From an engineering perspective, the agent abstraction can be suitably exploited for tackling cooperation of heterogeneous systems, facilitating semantic interoperability of independently developed software components. But, in order to support the sound design of infrastructures for agent systems, adequate models have to be studied as to grasp the important aspects of cooperation at the desired level of abstraction.

In this paper we focus on the semantics of Agent Communication Languages (ACLs), and study the approach based on the idea of describing an agent as a grey-box software component, representing its behaviour by means of transition systems. This framework provides an operational description of ACLs, considering the single-step evolution and interactive capability of an agent, which contrasts the classical frameworks based on intentional descriptions, which rely on the concept of agent mental state. Some examples are provided to show the flavours of the proposed model to describe various semantic aspects of communicative acts.

1 Motivation

Modern software systems become both increasingly large and more complex, moving towards more dynamic user interaction patterns. This has not just affected technical matters, but also the economical landscape of the Information Technology industry at large, creating a market where time between product conception and delivery shortens more and more. This pressure to release new features in shorter time raises very serious software engineering issues [12]. Individual productivity of software developers is quite hard to increase, so most organizations had to cope with the pressure by resorting to large-scale reuse politics.

As a result, most modern, feature-rich software systems are actually wrapping older, much less glamorous systems: while this approach is quite a cost effective one, it has one major drawback. As time passes and new parts are added, the conceptual integrity – that is, the property of being understandable through

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 111–125, 2003.

a coherent and possibly limited set of concepts – fades away and the system becomes less maintainable.

1.1 Homogenizing Heterogeneity: Modern Middleware and Agent Oriented Infrastructures

Looking at these premises, it is hardly surprising that all the most successful software infrastructure technologies of the last ten years explicitly targeted the integration of heterogeneous, distributed software applications. All the popular frameworks for software infrastructure, such as CORBA and Web Services, share the common trait of being a specification for component boundaries and interaction patterns, considering the component internals as an implementation specific part. All these technologies are meant to ease the integration of heterogeneous software systems without sacrificing the maintainability of the overall system. To reach this goal, they all inject conceptual integrity into heterogeneous software systems through a *model* that coherently encompasses the whole system. A relevant example of this approach is the Model Driven Architecture (MDA) [14] promoted by the Object Management Group.

An important role in this trend is played by the approach of *multi-agent systems*, which sees the whole system as composed by autonomous interacting agents, and introduces a higher description level using social notions to capture the behaviour of interactive systems [11]. A major aim of multi-agent systems is to enable software integration on a deeper level, namely shifting the integration process from *syntactic interoperability* to *semantic interoperability* [3]. This means producing a model and an infrastructure where independently developed components can interact by making assumptions on each other, that are perceived by human users as if the components can actually understand one another.

A relevant issue in the design of complex, distributed software systems concerns the level of detail of a model. Two common approaches to system modeling are *white box* modeling and *black box* modeling. In white box modeling, the system is described in term of its inner workings; this approach is troublesome to use in our case, because there is no single model that can coherently describe all the parts of an heterogeneous system with all their implementation details. Black box modeling, instead, only deals with the system at its boundaries, providing a description based solely on the external system behavior. Unfortunately, this approach has problems also, because it provides only small information about the system behaviour, thus poorly supporting the system design.

Therefore, a hybrid *grey box* modeling approach is adopted: the system is described within a coherent framework as a collection of interconnected parts, but abstracting away from the inner details of some these parts, according to the desired abstraction level.

In [17], the authors propose to apply a grey box modeling approach to the specification and design of multi-agent systems, focusing on the description of agent interactive aspects. This approach is quite sensible, because multi-agent systems are particularly geared towards highly heterogeneous distributed environments.

1.2 Semantics, Formalism and Software Development

In the field of agent systems, the problem of interoperability has in the last year faced the issue of formal semantics. The FIPA organization produced a comprehensive set of specifications for interoperable multi-agent systems, following a grey-box approach that describes software agents using a BDI framework [6] without actually requiring them to have an internal BDI architecture. In particular, semantics of communicative acts is formally described in terms of feasibility preconditions (FP) and rational effects (RE), expressed through a modal language with operators for modeling intentional aspects such as beliefs and intentions [8]. Having a formal semantics is often considered a strength of FIPA ACL, and it is cited as making it superior to more traditional approaches like distributed objects and even to other agent communication languages like KQML [2].

A common supporting statement for a formal semantics focuses on verification: if a software infrastructure has a formal semantics, an external authority can check implementations for compliance. However, among all the successful software infrastructures very few of them are currently provided with a formal semantics and check compliance with automated tools. Still, they all achieve software integration and interoperability, at least to some extent. Moreover, even when both formal definitions and automated checkers are available, sometime programmers do not really seem to care – as the HTML case shows.

However, formal semantics can really be useful during the design phase. While implementation is highly coherent, being defined precisely enough to be executed, its very low abstraction results in a myriad of details that quickly overwhelm the capabilities of formal methods. On the other hand, requirements and specifications can be very abstract, but they are also seldom coherent. In fact, coming from interaction with customers through a typically iterative process, requirements and specifications present a twofold incoherence: they are often logically contradictory (incoherence among different parts of the specification), and they are quickly evolving (incoherence between the specification at different time points). Only design strikes the right balance between abstraction and coherency, so that it can be effectively improved by a formal framework. During design, the designer does not deal with the real system, but with a simplified system model that allows reasoning without impairing creativity with too much detail. Moreover, though the design is supposed to produce a system satisfying customer requirements, the design model tends to be somewhat decoupled from the specific customer needs, and is thus more robust against requirements creeping. As a result, during the design phase formal semantics can be fruitfully used to check the coherence of the model – commonly referred to as its *soundness* – and to verify its properties of interest, enabling interception of errors in the early stages of the design and better guiding to correct implementations.

This does not mean that a formal semantics is useful to every software designer; rather, almost all of them can create meaningful or even insightful software without knowing any formal framework[1]. Though a formal semantics is directly useful only to a small subset of designers, its benefit can still be significant, because that small subset is exactly the one trying to cope with the hardest problems, often producing languages, tools and libraries that can be exploited by the development community at large. In the particular case of ACL semantics, which is the one here we are interested into, formal approaches may allow designers to precisely define the means by which an agent should send and receive messages, possibly supporting the design of the agent interface, the agent communicative acts, and negotiation protocols.

1.3 Overview

Given the background and motivation provided in this section, the remainder of the paper is organised as follows. Section 2 discusses various aspects concerning the design of agent-based systems, focusing on the desired features of a formal framework supporting it. Section 3 introduces the framework of transition systems on which our approach is based. Based on transition systems, in Section 4 we provide a framework for modeling agents, focusing on the operational description of their interactive behaviour. In Section 5 we go into details of defining ACL semantics by our framework, discussing aspects related to the management of FP and RE. Finally, in Section 6 we provide concluding remarks.

2 Enabling Sound Design of Multi-agent Systems

The main goal of this paper is trying to support multi-agent systems design with a formal framework. In order to pursue this goal, two broad classes of issues need to be addressed. First, the general nature of the design process has to be taken into account, and then the peculiarities of multi-agent systems can enter the picture.

Design is invention and creation, it is all about dynamism and tension towards a goal, so, a design rule should both describe a property of the design and suggest a roadmap to achieve it. This property is referred to as generativity in Christopher Alexander's work [1]; this building architect inspired not only his fellows, but also the software patterns community. From the mid-nineties the software pattern community treasured Alexander's ideas and gathered together a body of patterns of software development, making design experience from the past readily available to younger designers. However, when the application domain or the programming paradigm is new and relatively untested, the designer cannot count on patterns for generativity (at least, not completely) because there will not be much past design experience to leverage. Moreover, without a lot of

[1] It should be noted that, as a matter of fact, most designers hardly understand formal semantics and are not confident with their usages.

past history to feed his or her intuition, the designer will more likely appreciate a precise approach. Then, a formal framework aiming to support design will need the following features:

1. Being generative, that is combining description and dynamism into a calculus.
2. Focusing on fundamental issues, that can be solved once by a few skilled designers and leveraged many times from then on.

In the particular case of multi-agent systems design, the two properties above are not enough; others need to be added due to the specific intent of multi-agent systems, that is to obtain semantic interoperability. First of all, in the context of a specific application a shared ontology should be defined, describing entities from the domain of discourse so as to support a general understanding of the content of communicative acts. Then, an ACL semantics is to be defined in order to give an agreed upon representation of the motivations behind the observed speech act – e.g., the official semantics of FIPA ACL models agent communication using a BDI framework. The special emphasis put on semantic interoperability by the multi-agent systems approach suggests two more desirable properties for our formalism:

3. Supporting the description of autonomous software components, by allowing non-determinism in the system evolution and opaqueness with respect to the actual internal model of each agent.
4. Being integrated with the basic speech act communication model, where an agent can affect its environment both by performing direct actions on it and by uttering communicative acts.

These four properties guide us to the definition of a new framework for the description of an agent behaviour, which in the end we will use as a tool for formally defining an ACL semantics.

First of all, we intend to rely on an operational specification of an agent behaviour – describing its capability of performing interactions and changing its state – instead of taking the viewpoint of an agent as an entity with a mental state. While this approach generally lowers the abstraction level, it better supports the need for being generative of property 1. We achieve this result by relying on the framework of labelled transition systems, as shown in next sections.

Then, according to property 2 we will adopt a grey-box modeling approach for agents, focusing on the foundamental issue of the agent interactions, describing the agent part meant to manage the interactions with the environment while abstracting away from the agent internal details. Other papers [17, 18] discusses why this grey-box approach is suitable for tackling agent specific aspects as requested by properties 3 and 4.

3 The Framework of Transition Systems

In the remainder of the paper we adopt the following syntactical notations. Given any set X, this is automatically ranged over by variables x, x', x'', x_0, and so on – and analogously Y is ranged over by y, y', etcetera, and similarly for any set. The set of multisets over X is denoted by $\overline{X}$, and is ranged over by variable $\overline{x}$, the set of sequences over X by X^*, ranged over by x^*. Union of multisets $\overline{x}$ and $\overline{x}'$ is denoted by $\overline{x}|\overline{x}'$, concatenation of sequences x_0^* and x_1^* by $x_0^*; x_1^*$, void multiset and empty sequence by $\bullet$. Given any set X, symbol $\bot_X$ is used to denote an exception value in set $X_\bot$ defined as $X \cup \{\bot_X\}$. Variable $x_\bot$ is automatically defined that ranges over set $X_\bot$.

The semantic approach we describe in this paper is based on the framework of *(labelled) transition systems* [9, 16], which is widely used in the field of concurrency theory to provide an operational semantics to process algebras [4], so as to specify the interactive behaviour of systems. A transition system over set X is a triple $\langle X, \longrightarrow, Act\rangle$ where X is the set of states of the system of interest, Act is called the set of actions, and $\longrightarrow \subseteq X \times Act \times X$ is a relation. Let $x, x' \in X$, and $act \in Act$, $\langle x, act, x'\rangle \in \longrightarrow$ is written $x \xrightarrow{act} x'$ for short, and means that the software component of interest may move from state x to state x' by way of action act. Once the set of actions is specified and the structure of set X is set up, the content of transition relation $\longrightarrow$ can be intensionally given in terms of a set of rules of the kind:

$$\frac{\phi_{\mathbb{B}}^0 \qquad \phi_{\mathbb{B}}^1 \qquad \ldots \qquad \phi_{\mathbb{B}}^n}{\phi_X \xrightarrow{\phi_{Act}} \phi'_X}$$

where, given any set S, ϕ_S is any formula (possibly involving free variables) whose ground terms range over S – e.g., the upside part of the rule contains a number of predicates on the free variables, in that they are formulae yielding a boolean value in $\mathbb{B}$. Given a number of such rules, $x \xrightarrow{act} x'$ is considered to hold if for at least one rule ρ and one substitution σ of the free variables to ground terms, the downside part of ρ gets substituted by σ to $x \xrightarrow{act} x'$ and *all* predicates in the upside part of ρ are satisfied by the application of σ. In the case zero predicates needs to be specified, only the downside part of the rule is actually reported.

Actions can be given different interpretations. In the most abstract setting, actions are meant to provide a high-level description of a system evolution, abstracting away from details of the inner system state change. In the sequence of transitions $x_0 \xrightarrow{a_0} x_1 \xrightarrow{a_1} x_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} x_n$, the system evolution characterised by states $x_0, x_1, \ldots x_n$ is associated to the action sequence $a_0, \ldots, a_{n-1}$, which can be though of as an abstract view of that evolution. In a sense, actions can be used to describe what an external, virtual entity is allowed to perceive of the system evolution, generally providing only a partial description.

Transition systems feature interesting properties that make them particularly appealing as mathematical structures for describing the behaviour of interactive systems, in which case actions are used to represent the system's interactions with the environment. Most importantly, transition systems intrinsically take into account non-determinism, which is a peculiar aspect of interactive systems, especially as far as asynchrony is concerned. For instance, in a given transition relation both the triples $\langle x, a, x' \rangle$, $\langle x, a', x'' \rangle$, and $\langle x, a', x''' \rangle$ can occur – meaning that in state x, the system may either perform action a or a', and moreover, by performing a' it can either move to x'' or x'''. Non-determinism is generally used as a powerful abstraction tool, in that the specification of a system actually provides a number of allowed behaviours. Several remarkable concepts related to non-determinism are promoted by transition systems, such as the notions of system *observation* and *refinement* of specifications [13], which are here briefly discussed.

Since actions in a transition system can be interpreted as a view of the system transition from a state to another, it is then possible to characterise a whole system evolution in terms of aspects such as the actions permitted at each step and the actions actually executed. This characterisation is made according to a given *observation semantics*, associating to each system's evolution a mathematical description called *observation*. For instance, according to the so-called *trace semantics* [9, 5], an observation is simply made by one of the (possibly infinite) sequences of actions executed, also called *trace histories*. By non-determinism, generally more observations are possible for a system – either because of the environment interacting in different ways, or because of different internal behaviours –, so that according to the given semantics the system itself can be characterised by the set of all its possible observations.

The notion of *refinement* then naturally comes in: a system description is considered an "implementation" of another – i.e., it is more directly executable – if it is more deterministic, allowing for strictly fewer observations [9]. In this framework, transition systems can be generally exploited as a tool for either describing system abstract specifications and system implementations. Besides proving conformance via standard techniques such as algebraic verification [10], this framework allows us to describe system specifications and to derive system implementations from them.

4 The Operational Specification of an ACL Semantics

The semantics of agent communication languages such as FIPA ACL is expressed by defining preconditions and expected effects of each performative, that is, by describing the conditions for sending and the effect of receiving communicative acts. This amounts to describe the behaviour of the agent part in charge of managing communications with the environment – which we call the *agent interface*. That agent part specifies how message reception and sending is related to the agent internal part, which FIPA ACL specification understands in terms of the notion of mental state. This technique relies on a grey-box approach, where agent internal aspects and dynamics are abstracted away, while only the part responsible for managing communication with the environment is represented.

Our goal here is to generalize this approach, by providing an abstract framework where representing the relationship between agent communications and changes to the agent internal part, the latter represented in an abstract way instead of relying on mental properties. Since this model concerns the interactive aspects of software components we naturally rely on the framework of transition systems, in contrasts to the usual framework based on modal logics for the description of mental states and their evolution.

4.1 A Transition Systems Semantics for Grey-Boxes

Our intention here is to represent an agent in terms of a software abstraction interacting with its environment in one of two ways: *(i)* by a *reactive action* the agent receives a communicative act, changes its internal state, and possibly reactively sends a communicative act; *(ii)* by a *proactive action* the agent proactively changes state and then possibly sends a communicative act.

Following the grey-box approach, then, we mean to explicitly represent only the part of the agent devoted to managing these interactions with the environment, which we call the *(agent) core*, asbtracting away from the part dealing with agent internal issues, called the *(agent) internal machinery.* The set of states of the agent core part, called *observable states*, are here supposed to range over set O: they represent the subpart of the agent status which directly influence interactions. In order to model the effects of the internal machinery on the agent core, which is responsible for the execution of proactive actions, we introduce the set of *(internal) events* E. Raising an internal event is to be understood as the result of a (possibly complex) computation within the internal machinery whose final effect is meant to influence the agent interactive behaviour – e.g., when the effect of planning is to send a message outside. On the other hand, the agent interactions with the environment typically influence the agent internal behaviour, so we suppose that each time an action (either a reactive or proactive) is processed the agent core performs an update to the internal machinery. Each update is represented by a term ranging over set $U_\perp$ (supposed to be disjoint from E), where update $\perp_U$ means that no change is currently significant for the internal machinery.

So, the behaviour of the agent core can be understood in terms of a transition system $\mathcal{O} = \langle O, \longrightarrow_{\mathcal{O}}, Act_O \rangle$, describing how the observable state evolves as actions are executed. In particular, actions in Act_O are divided into reactive actions Act^r_O and proactive actions Act^p_O, defined by $Act^r_O ::= ic \triangleright u_\perp \triangleright oc_\perp$ and $Act^p_O ::= e \diamond u_\perp \diamond oc_\perp$. Transitions in $\mathcal{O}$ are then of the kind:

$$o \xrightarrow{ic \triangleright u_\perp \triangleright oc_\perp}_{\mathcal{O}} o' \qquad o \xrightarrow{e \diamond u_\perp \diamond oc_\perp}_{\mathcal{O}} o'$$

The former represents the reactive action where observable state o moves to o' by receiving act ic, performing update $u_\perp$, and then sending act $oc_\perp$ – the case where the latter is $\perp_{Oc}$ means that no output act will be sent. Analogously, the latter represents the proactive action where o moves to o' by intercepting internal event e, performing update $u_\perp$, and sending act $oc_\perp$ (again possibly void). So, given an ACL whose input communicative acts range in Ic and output communicative acts range in Oc (which are typically the same set), its formal semantics can be simply given by transition system $\mathcal{O}$ describing in an abstract way the behaviour of the agent interface responsible for sending acts Oc and receiving acts Ic accordingly to the ACL semantics.

Indeed, by non-determinism, it is possible to characterise the behaviour of an agent complying to this semantics by simply assuming the most abstract characterisation for the internal machinery, that is, supposing that at any time any internal event can be raised and any update is accepted. This leads us to the transition system $\mathcal{S} = \langle O, \longrightarrow_{\mathcal{S}}, Act_S \rangle$, with $Act_S = Ic_\perp \times Oc_\perp$ and $\longrightarrow_{\mathcal{S}}$ defined by rules:

$$\frac{o \xrightarrow{ic \rhd u_\perp \rhd oc_\perp}_{\mathcal{O}} o'}{o \xrightarrow{\langle ic, oc_\perp \rangle}_{\mathcal{S}} o'} \qquad\qquad \frac{o \xrightarrow{e \diamond u_\perp \diamond oc_\perp}_{\mathcal{O}} o'}{o \xrightarrow{\langle \perp_{Ic}, oc_\perp \rangle}_{\mathcal{S}} o'}$$

Transition $o \xrightarrow{\langle ic_\perp, oc_\perp \rangle}_{\mathcal{S}} o'$ means that the agent move from core state o to o' by receiving input $ic_\perp$ (or proactively, if this is $\perp_{Ic}$) and by sending $oc_\perp$ (or nothing, if this is $\perp_{Oc}$). The case where $ic_\perp =\perp_{Ic}$ or not distinguishes proactive actions from reactive actions, respectively. From an abstract point of view, transition system $\mathcal{S}$ can be considered as the composition $\mathcal{O} \otimes \mathcal{G}$ of the agent core $\mathcal{O}$ specifying the ACL semantics, and a general internal machinery $\mathcal{G}$. $\mathcal{S}$ can be considered as the specification of the most general behaviour of an agent complying to the ACL semantics described by $\mathcal{O}$.

On the other hand, an actual implementation of this specification – that is, the implementation of a compliant agent – can be defined by adding to $\mathcal{O}$ the description of an actual, specific agent internal machinery. The internal machinery part can be specified as a component raising internal events and consuming updates, that is, as a transition system $\mathcal{I} = \langle I, \longrightarrow_{\mathcal{I}}, E \cup U_\perp \rangle$. A transition $i \xrightarrow{e}_{\mathcal{I}} i'$ represents the internal machinery raising event e and correspondingly moving from state i to i', whereas transition $i \xrightarrow{u_\perp}_{\mathcal{I}} i'$ represents the internal machinery moving from i to i' due to update $u_\perp$ (here we suppose that $i \xrightarrow{\perp_U}_{\mathcal{I}} i$ always holds). So, by coupling the specification of the core $\mathcal{O}$ – representing the ACL semantics – to the specification of the internal machinery $\mathcal{I}$ – representing the agent peculiar aspects – the specification of a realisation $\mathcal{R} = \mathcal{O} \otimes \mathcal{I}$ of the agent is obtained. In particular this is a transition system $\mathcal{R} = \langle O \times I, \longrightarrow_{\mathcal{R}}, Act_S \rangle$, with same actions as $\mathcal{S}$, defined by rules:

$$\frac{o \xrightarrow{ic \rhd u_\perp \rhd oc_\perp}_{\mathcal{O}} o' \qquad i \xrightarrow{u_\perp}_{\mathcal{I}} i'}{\langle o, i \rangle \xrightarrow{\langle ic, oc_\perp \rangle}_{\mathcal{R}} \langle o', i' \rangle} \quad \frac{i \xrightarrow{e}_{\mathcal{I}} i' \quad o \xrightarrow{e \diamond u_\perp \diamond oc_\perp}_{\mathcal{O}} o' \quad i' \xrightarrow{u_\perp}_{\mathcal{I}} i''}{\langle o, i \rangle \xrightarrow{\langle \perp_{Ic}, oc_\perp \rangle}_{\mathcal{R}} \langle o', i'' \rangle}$$

The core property validating our methodological approach to the semantics of ACLs states that composition operator $\otimes$ is compatible with respect to the chosen observation semantics, which implies that if $\mathcal{I}$ is a refinement of $\mathcal{G}$, than $\mathcal{O} \otimes \mathcal{I}$ is a refinement of $\mathcal{O} \times \mathcal{G}$ [5]:

$$\mathcal{I} \preceq \mathcal{G} \quad \Longrightarrow \quad \mathcal{O} \otimes \mathcal{I} \preceq \mathcal{O} \times \mathcal{G}$$

This property means that an agent obtained by using $\mathcal{O}$ as a wrapper for an actual internal machinery $\mathcal{I}$ is a refinement of $\mathcal{S}$, that is, complies to the ACL semantics described by $\mathcal{O}$. Theorem 1 shows that this property is simply proved in the case we choose trace semantics as observation semantics. Using other observation semantics such as bisimulation can be more appropriate depending on the desired compositionality properties required for agents [9]. Deepening this technical issue is left as future work.

Theorem 1. *For any $o_0 \in O$ and $i_0 \in I$, $traces_{\mathcal{R}}(\langle o_0, i_0 \rangle) \subseteq traces_{\mathcal{S}}(o_0)$, i.e.:*

$$\{(a_0; a_1; \ldots; a_n) \in Act_S^* : \langle o_j, i_j \rangle \xrightarrow{a_j}_{\mathcal{R}} \langle o_{j+1}, i_{j+1} \rangle, 0 \leq j < n\} \subseteq$$
$$\{(a_0; a_1; \ldots; a_n) \in Act_S^* : o_j \xrightarrow{a_j}_{\mathcal{S}} o_{j+1}, 0 \leq j < n\}$$

Proof sketch. Suffice it to prove that for any $o \in O$, $i \in I$, and $a \in Act_S$, when transition $\langle o, i \rangle \xrightarrow{a}_{\mathcal{R}} \langle o', i' \rangle$ occurs then we have also $o \xrightarrow{a}_{\mathcal{S}} o'$, which is true by definition of $\rightarrow_{\mathcal{R}}$ and $\rightarrow_{\mathcal{S}}$.

4.2 Intentional vs. Operational Specification

We refer to the specification of an ACL semantics in terms of transition system $\mathcal{S}$ as an *operational specification*. This name comes from the typical usage of transition systems as operational semantics for process algebras, and generally amounts to specifications describing the single-step execution and interaction capabilities of systems. In the context of ACLs, our specification describes the allowed communicative acts sent and received by a compliant agent, and their relations with the agent core O and with internal events and updates E, U – the two latter describing the mutual effects of interactions and internal behaviour.

In contrast to this approach, existing ACL semantics such as FIPA ACL adopt what we may call an *intentional specification*. In fact, they endorse the notion of *intentional stance* [7], describing the agent behaviour in terms of ascriptions to the system of beliefs, desires, and intentions, that is, in terms of the notion of mental state.

These two approaches are not completely different, e.g. they both rely on a grey-box modeling approach, even though they tackle complementary aspects. On the one hand, the operational specification abstracts away from the "meaning" of the represented agent status, whereas the intentional specification understands it in terms of mental properties. On the other hand, the operational specification sticks to the interactive aspects of the agent behaviour, thus describes the whole set of allowed interactive behaviours for the agent – enjoying the features of non-determinism. No hypothesis can be made on which actual behaviour will be chosen, whereas intentional specifications provide a means for trying to explain and predict agent choices, namely, by hypothesising that the agent mental state evolves in a *rational* way.

5 Communicative Acts Semantics

In this section we describe with some further detail how our approach can be exploited for describing the semantics of communicative acts.

5.1 Unspecified Semantics

As first example, we consider the trivial case where no actual specification is associated to communicative acts, so that no relevant information can be actually inferred by the agent interaction history. This simple example should be considered as an introduction to our notation and methodology. We suppose that reception of communicative acts and their processing by the internal machinery are decoupled. We define $O = Ic^*$: at any time, the observable status only contains the sequence of input acts received but not yet processed (namely, the queue of incoming requests). Internal events are of two kinds: *(i)* get represents a request for the internal machinery to consume the next input act, and *(ii)* $send(oc)$, represents the internal machinery producing the output act oc to be sent out. Updates are terms $prc(ic)$, communicating to the internal machinery the new incoming act ic to be processed.

So, the operational semantics of our ACL is simply defined by the three rules:

$$ic^* \xrightarrow{ic \rhd \bot_U \rhd \bot_{Oc}}_{\mathcal{O}} ic; ic^* \quad ic^*; ic \xrightarrow{get \diamond prc(ic) \diamond \bot_{Oc}}_{\mathcal{O}} ic^* \quad ic^* \xrightarrow{send(oc) \diamond \bot_U \diamond oc}_{\mathcal{O}} ic^*$$

They respectively represents: *(i)* incoming act ic being received and then stored into the observable state, *(ii)* event get causing the first message ic of the queue to be processed, represented by update $prc(ic)$, and *(iii)* event $send(oc)$ causing communicative act oc to be sent out.

5.2 Modeling Feasibility Preconditions

In FIPA ACL semantics, some FP and RE are associated to any communicative act. However, RE are not mandatory for the receiver: they can be just used by the sender so as to hypothesise the effect of sending a message, which is used to generally figure out plans to achieve goals. On the other hand, an agent can emit an act only if its FP are satisfied. We can take into account this constraint by considering $O = Ic^* \times \overline{Oc} \times M$ as a triple respectively containing the queue of input acts to be processed, the (multi-)set of output acts whose FP are currently satisfied, and the state M based on which FP are calculated (e.g. the mental state in FIPA ACL semantics). We suppose function $\mathtt{fp} \in M \mapsto \overline{Oc}$ takes elements in M and yields the corresponding set of output acts whose FP are satisfied.

Internal events are now of three kinds: besides events get and $send(oc)$ as in previous case, event $move(m')$ represents the state M of the agent changing to new value m' – modeling e.g. a belief update. Then, we also add to $prc(ic)$ the new kind of update $fea(\overline{oc})$, used to communicate to the internal machinery the new output acts whose FP are satisfied. Operational semantics is built so that function `fp` is computed each time component M changes, that is:

$$\begin{array}{c}
\langle ic^*, \overline{oc}, m\rangle \xrightarrow{ic \rhd \bot_U \rhd \bot_{Oc}}_{\mathcal{O}} \langle ic; ic^*, \overline{oc}, m\rangle \\
\langle ic^*; ic, \overline{oc}, m\rangle \xrightarrow{get \diamond prc(ic) \diamond \bot_{Oc}}_{\mathcal{O}} \langle ic^*, \overline{oc}, m\rangle \\
\langle ic^*, \overline{oc}, m\rangle \xrightarrow{move(m') \diamond fea(\mathtt{fp}(m')) \diamond \bot_{Oc}}_{\mathcal{O}} \langle ic^*, \mathtt{fp}(m'), m'\rangle \\
\langle ic^*, oc|\overline{oc}, m\rangle \xrightarrow{send(oc) \diamond \bot_U \diamond oc}_{\mathcal{O}} \langle ic^*, oc|\overline{oc}, m\rangle
\end{array}$$

The former two rules are analogous to the previous case. The third rule represents the agent proactively updating its state M from value m to value m', by which FP are recomputed affecting the set of admissible output acts and correspondingly performing an update. The fourth rule describes the agent proactively sending one of the output events whose FP are satisfied.

5.3 On Rational Effects

Whereas FIPA ACL semantics promotes full autonomy for agents, and so it mandates no RE to their communicative acts, here it is interesting to consider also the case where an agent processing a message cause some RE to be applied. First of all, a function $\mathtt{re} \in M \times Ic \mapsto M$ has to be defined that takes the current state M and the input act to be processed, and yields the new state M after applying the act's RE. The proactive action corresponding to the processing of a new input act is then of the kind:

$$\langle ic^*; ic, \overline{oc}, m\rangle \xrightarrow{get \diamond prc(ic) \diamond \bot_{Oc}}_{\mathcal{O}} \langle ic^*, \overline{oc}, \mathtt{re}(m, ic)\rangle$$

Both consuming act ic and applying its RE to state M.

5.4 Conversation-Based Semantics

As discussed in [15], one of the main drawbacks of current semantics of FIPA ACL is its lack of the concept of conversation protocol. Indeed, agents can cooperate by exploiting well-defined interaction protocols, where a statically fixed sequence of communicative acts has to be realised in order to achieve the intended aim – for instance, negotiating an auction. Whereas FIPA ACL includes protocol-oriented performatives, the burden of setting up the proper interactions protocols is left unspecified: the corresponding management is meant to be conceptually located within the internal machinery.

In our framework, we can add the notion of a conversation to the semantics of communicative acts, by taking it into account in the specification of an agent core. Formal details are omitted here for brevity, whereas only the basic intuition about the idea is provided. An interaction protocol can be considered as a sequence of communicative acts; at a given time an agent can participate to a number of such protocols. A conversation is an instance of a protocol, which keeps track of the communicative acts already occurred: the agent core can be designed so as to store all the information about the conversations the agent is participating on. As a result, the requirements an agent has to satisfy in order to participate to a protocol – in terms of the allowed sequence of communicative acts – can be directly represented in the agent core. By moving this specification from the agent internal machinery to agent core, the task of ensuring compliance of a conversation with respect to a protocol is generally facilitated.

5.5 From Specification to Implementation

The specification of an ACL semantics provided by our framework enjoys a very appealing property: it allows for the automatic generation of a wrapper agent complying with the semantics. [2] This wrapper has eventually to be completed with an implementation of the actual agent internal behaviour, which can be built using any architecture and implementation technique. For instance, the wrapper may actually hides a an internal BDI machinery, constraining its deliberation process.

Our framework can be fruitfully applied also when the internal behaviour is not implemented by the BDI framework, but for instance through a simple active object implementing a daemon agent. In spite of its simplicity, our wrapper would make this agent cooperate with existing, possibly smart agents in a compliant way with respect to the ACL semantics, thus enjoying the services these agents may be able to offer. In the case of legacy systems, on the other hand, the compliant wrapper may be used as a proper interface for the cooperation with other agents of the MAS, while only an intermediate part remains to be designed to fill the gap between the wrapper and the legacy system.

6 Conclusions

In this paper we study an approach to the semantics of ACLs based on transitions systems, which we refer to as the operational specification of ACLs. By our framework, we mean to strengthen the cooperation of agents built over different actual architectures: in particular, our aim here is to allow simple agents of a MAS to benefit from the cooperation with smart agents – e.g. implementing intelligent behaviour through a BDI framework. In particular, our approach focuses on the interactive behaviour of agents and abstracts away from the concept of mental state on which most existing ACLs are based.

The framework provided here is a generalisation of the approach presented in [18], which is based on the *observation framework* for agents [17]. There, instead of describing the issue of ACL semantics in general as developed here, a particular abstract architecture is provided based on an ontology for the *observation* issue. This abstract architecture is claimed to be more widely applicabile to different agent actual architectures, because it captures the very notion of

[2] Notice that transition system semantics are well recognised as suitable tools for directly leading to implementations, in that their specification is operational.

agent interaction with the environment, and the idea of an agent allowing its status to be observed and altered from other entities. On the other hand, the generalised framework presented here simply sticks to idea of grey-box specifications for agents, so it better allows us to foster comparison with existing ACL semantics.

One of the main future works of this research is trying to adopt the typical proof techniques for process algebras – which exploit transition systems semantics – to the domain of ACLs, providing relevant tools by which designers can enforce compliance of agent designs and agent interaction protocols, so as to strengthen semantics interoperability in multi-agent systems.

References

[1] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979. 114

[2] ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group, July 1993, July 1993. 113

[3] Federico Bergenti. On Agentware: Ruminations on why we should use agents. In *Engineering Societies in the Agents World*. Springer-Verlag, 2002. 112

[4] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001. 116, 124

[5] Manfred Broy and Ernst-Rüdiger Olderog. Trace-oriented models of concurrency. In Bergstra et al. [4], chapter 2, pages 101–195. 117, 119

[6] Philip R. Cohen and H.J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–361, 1990. 113

[7] Daniel Dennett. *The Intentional Stance*. MIT Press, Cambridge, MA, 1987. 120

[8] FIPA. FIPA communicative act library specification. `http://www.fipa.org`, 2000. Doc. XC00037H. 113

[9] R.J. van Glabbeek. The linear time – branching time spectrum I. The semantics of concrete, sequential processes. In Bergstra et al. [4], chapter 1, pages 3–100. 116, 117, 120

[10] J.F. Groote and M.A. Reniers. Algebraic process verification. In Bergstra et al. [4], chapter 17, pages 1151–1208. 117

[11] N. R. Jennings and J. R. Campos. Towards a social level characterisation of socially responsible agents. *IEEE Proceedings on Software Engineering*, 144(1), 1997. 112

[12] Steve McConnell, Donald Reifer, John Favaro, and Martin Fowler. Engineering internet software. *IEEE Software*, 19(2), 2002. 111

[13] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989. 117

[14] OMG. Model Driven Architecture. `http://www.omg.org/mda`, 2002. 112

[15] Jeremy Pitt and Ebrahim Mamdani. A protocol-based semantics for an agent communication language. In *16th Intl. Joint Conf. on Artificial Intelligence (IJ-CAI '99)*, pages 486–491, Stockholm, Sweden, 1999. Morgan Kaufmann. 122

[16] Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Department of Computer Science, AArhus University, 1991. 116

[17] Mirko Viroli and Andrea Omicini. Modelling agents as observable sources. *Journal of Universal Computer Science*, 8, 2002. 112, 115, 123

[18] Mirko Viroli and Andrea Omicini. Towards an alternative semantics for FIPA ACL. In *Cybernetics and Systems 2002*, volume 2, pages 689–694, Vienna, Austria, 2002. Austrian Society for Cybernetic Studies. 115, 123

Access-as-you-need: A Computational Logic Framework for Accessing Resources in Artificial Societies

Francesca Toni[1] and Kostas Stathis[2]

[1] Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK,
ft@doc.ic.ac.uk, http://doc.ic.ac.uk/~ft

[2] Department of Computing, School of Informatics, City University London,
Northampton Square, London EC1V 0HB, UK,
k.stathis@city.ac.uk,
http://soi.city.ac.uk/~kostas

Abstract. We investigate the application of abductive logic programming, an existing framework for knowledge representation and reasoning, for specifying the knowledge and behaviour of software agents that need to access resources in a global computing environment. The framework allows agents that need resources to join artificial societies where those resources are available. We show how to endow agents with the capability of becoming and ceasing to be members of societies, for different categories of artificial agent societies, and of requesting and being given or denied resources within societies. The strength of our formulation lies in combining the modelling and the computational properties of abductive logic programming for dealing with the issues arising in resource access within artificial agent societies.

1 Introduction

Global computing [5] is a new vision for information and communication technologies whereby most physical objects that people use will be transformed into electronic devices with computing processors and embedded software designed to perform (or control) a multitude of tasks in people's everyday activities. Many of these devices will also be able to communicate with each other and to interact with the environment in carrying out the tasks they are designed for. The expectation is that the design of computational and information processing systems will rely entirely on the available infrastructure and processing power around us. The challenge this vision poses is huge in that it requires the definition and exploitation of dynamically configured applications of software entities interacting in novel ways with other entities to achieve or control their computational tasks.

Global computing environments are intended to be highly *open and dynamic*: physical devices will be mobile or portable, connectivity and bandwidth will not be constant, computational processes and data would need to migrate, and

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 126–140, 2003.

applications will come and go in the environment the same way people who use them will. In such unpredictable and open environments one of the main issues is how to control the availability of the resources that are active in an application at any given time.

From our involvement in the SOCS project[15], we are motivated by a *travelling businessman scenario* presented in [16]. This scenario can be summarised as follows. A businessman travels for work purposes to a foreign country and, in order to make his trip easier, carries a personal communicator, namely a device that is a hybrid between a mobile phone and a PDA. The application running on this personal communicator provides the environment for a personal service agent [8], namely a piece of software that augments the direct manipulation interface of the device with implicit management. By implicit management we mean that the agent provides proactive information management within the device [19] and flexible connectivity to smart services available in the global computing environment the businessman travels within. We interpret connectivity to smart services as *access to resources*.

We assume that resources are available within and managed by *artificial agent societies*[9] in the global computing environment. Examples of such societies in the travelling businessman scenario are (the society of agents supporting) an airport, a hotel, a train station.

In SOCS [15], we use the term agent society simply to refer to a possibly complex organisation of agents that supports and complements the activities of a physical, human society, in a global computing environment. In particular, artificial societies complement physical societies by being composed of agents that act on behalf of people (whether individuals or groups) in order to access, manage, control and authorise the use of resources required by global computing applications. Artificial societies can be understood as systems being composed of autonomous computational entities where

- activity is not centrally controlled, either because global control is impossible or impractical, or because the entities are created or controlled by different owners.
- the configuration of the system varies over time; for instance, the system is open to the introduction of new computational entities and the departure of already existing ones.

We do not regard artificial societies merely as locations where resources can be found. Although artificial societies are accessible from physical locations, they are further characterised by sets of rules and conventions that govern the accessibility, management, control and authorisation to use resources, as we will see later on.

We are particularly interested in investigating the situation whereby an agent, who is in pursuit of its goals in order to satisfy the needs of a person (or group) in a human society, becomes a *member* of one or more artificial societies to obtain resources that are required by the person (or group). We follow the categorisation of artificial societies given in [1], whereby societies may be *open*, with no eligibility requirements for members-to-be, *closed*, with no agent not already part of these societies being allowed to enter, *semi-open*, with agents allowed to become members if they satisfy some eligibility criteria, and *semi-closed*, with agents not allowed to become members under any circumstances, but allowed, if eligible, to have a representative created or nominated within the society, to act on their behalf.

In this paper, we provide a knowledge representation and reasoning framework based on *abductive logic programming* [6] for specifying the knowledge and behaviour of agents joining and leaving societies according to the resources they need in order to satisfy their goals. We rely upon the computational properties of abductive proof procedures that abductive logic programming is equipped with, to reason with the knowledge of agents and allow agents to exhibit the required behaviour. The execution model of an agent uses the proof procedure within an observe-reason-act cycle that interleaves the execution of the proof procedure with observation and action execution [7, 12, 13, 14].

One of the advantages of our approach is that the specifications that we develop are executable, so that our approach could be used to actually build artificial societies. At the same time, our approach does not exclude the proposed specifications to be implemented in a different framework, as long as the operational semantics of the proof procedure is maintained. Thus, our framework can be seen as a logical model of the required agent behaviour. In this sense, societies in our framework are open to agents built in different frameworks, as soon as they conform to the specified behaviour, in the same way agents are required to conform to communication protocols.

Another advantage of our approach is that it lends itself to a formal analysis (such as the one in [13, 14]) of the multi-agent systems and artificial societies that are designed or built according to our approach. Such analysis, however, is beyond the scope of this paper.

The structure of the paper is as follows. In section 2 we present the framework of abductive logic programming for modelling agents' beliefs and behaviour. In section 3 we address the problem of how resources are requested and how requests are processed. In section 4 we specify how agents might join or leave societies to obtain resources, and how societies admit/reject new members or expel existing ones. We concentrate on open, semi-open and semi-closed societies (we disregard closed societies as they do not allow for dynamic entry and are thus uninteresting for our purposes). In section 5 we exemplify the computational behaviour of agents in our framework, by means of a simple instance of the travelling businessman scenario outlined above. Section 6 concludes.

2 Abductive Logic Programming

Abductive logic programming is a general-purpose knowledge representation and reasoning framework that can be used for a number of applications [6, 7, 3, 11, 2, 12, 13, 14, 20]. It relies upon a symbolic, logic-based representation of the beliefs of agents, for any given domain of application, via abductive logic programs, and the execution of logic-based reasoning engines, called abductive proof procedures, for reasoning with such representations. *Abductive logic programs* consist of

- A *logic program*, P, namely a set of if-rules of the form $Head \leftarrow Body$, where $Head$ is an atom, $Body$ is either $true$ (in which case we write the if-rule simply as $Head$) or a conjunction of (negations of) atoms. All variables in $Head$ and $Body$ are implicitly universally quantified from the outside. is understood as a (possibly incomplete) set of beliefs by the agent.
- A set of *integrity constraints*, I, namely if-then-rules of the form $Condition \Rightarrow Conclusion$, where $Condition$ is either $true$ (in which case we write the if-then-rule simply as $Conclusion$) or a conjunction of (negations of) atoms, $Conclusion$ is $false$ or a (possibly existentially quantified) conjunction of atoms, and all variables in $Condition$ are implicitly universally quantified from the outside. The integrity constraints are understood as properties that must be "satisfied" by any extension of the logic program by means of atoms of abducible predicates, in the same way integrity constraints in databases are understood as properties that must be "satisfied" in every database state.
- A set of *abducible predicates*, Ab, namely predicates occurring in the logic program and the integrity constraints but not in the $Head$ of any if-rule.

In this paper, for convenience of presentation, we allow for $findall$ and $forall$ conjuncts, understood as in PROLOG, to occur in the $Body$ of if-rules in P. Also, the set of abducible predicates Ab of an agent is structured as follows: $Ab = Actions \cup Observables$, such that $Actions \cap Observables = \emptyset$, where $Actions$ represent actions that the agent can execute and $Observables$ represent phenomena that the agent can observe. In general, the $Actions$ can be either physical or communicative, namely utterances that the agent can perform. In this paper, all abducibles of all agents we consider are communicative. The abducibles for sections 3-5 are:

$\mathbf{join}(Agent, Soc, Time)$, $\mathbf{leave}(Agent, Soc, Time)$,
$\mathbf{request}(Agent, Soc, Resource, Time)$,
$\mathbf{admit}(Auth, Agent, Soc, Time)$, $\mathbf{deny}(Auth, Agent, Soc, Time)$,
$\mathbf{represent}(Auth, Agent, Proxy, Soc, Time)$,
$\mathbf{assign}(Auth, Proxy, Agent, Soc, Time)$, $\mathbf{expell}(Auth, Agent, Soc, Time)$,
$\mathbf{broadcast\_joined}(Auth, Soc, Agent, Time)$,
$\mathbf{broadcast\_left}(Auth, Soc, Agent, Time)$,
$\mathbf{accept}(Controller, Agent, Soc, Resource, Time)$,
$\mathbf{refuse}(Controller, Agent, Soc, Resource, Time)$,

The meaning of these utterances will become clear later. Here, note that the first (second) argument of every such utterance is the utterer (receiver, resp.). Utterances with the agent as utterer (receiver) are actions (observables, resp.).

Given an abductive logic program (P, I, Ab) and an *observation* or *goal* (i.e. a possibly empty conjunction of literals) Q, the task of an *abductive proof procedure* is to compute a so-called *explanation* for Q, namely a pair (E, θ) consisting of a (possibly empty) set E of atoms in abducible predicates in Ab and a (possibly empty) variable substitution θ for the variables in Q, such that $P \cup E$ *entails*

$Q\,\theta$ and $P \cup E$ *satisfies* I. Various notions of entailment and satisfaction can be adopted, for example entailment and satisfaction could be entailment and consistency, respectively, in first-order, classical logic. However, the provision of such notions is beyond the scope of this paper, see [6] for a detailed discussion.

Examples of abductive proof procedures can be found in [6, 4, 11]. Typically, abductive proof procedures interleave *backward reasoning* (also called *unfolding*) with if-rules in the logic program (reduction of $Head$ to $Body$) and *forward reasoning* (also called *propagation*) with if-then-rules in the integrity constraints (addition of $Conclusion$ to $Condition$). Backward reasoning with the if-rules allows to generate the kernel of candidate explanations, whereas forward reasoning with the if-then-rules allows to rule out unwanted candidate explanations and expand the kernel of promising ones. Indeed, some kernel explanations might allow to explain the observation but might fail to take into account the integrity constraints, whereas others might need to be augmented in order to do so. The interleaving terminates when no more backward or forward reasoning can be performed in a branch in the tree of candidate explanations: the explanation can then be extracted from the branch.

While modelling agents via abductive logic programming, an abductive proof procedure is executed within an *observe–reason–act* cycle that allows the agent to be alert to the environment and react to it as well as reason and devise plans [7, 14]. This cycle interleaves the execution of the abductive proof procedure (*reason*) for the computation of explanations for observations and goals, the perception of inputs from the environment, to be added dynamically to the observation or goal to be explained (*observe*) and the selection of actions (i.e. abducibles in explanations) to be performed in the environment (*act*). Note that, in order for the agents to be reactive to the environment in which they are situated, and thus to the inputs from the agents that reside in the environment, agents might need to select actions and perform them before the computation of explanations is finalised. In this paper we will refer to the interleaved execution of an abductive proof procedure as illustrated above, as the *execution model* of the agent. We will assume that the observation of observables (utterances from other agents) is left to the execution model, and that when executing an utterance, its time argument is instantiated to the current time.

In the next two sections, we give a concrete abductive logic program modelling the knowledge of agents that need to access resources and might decide to join societies in order to gain access, and, in section 5, we use this modelling to illustrate the behaviour of the abductive proof procedure of [11] for an example within the travelling businessman scenario. Here, we illustrate by means of a simple example the use of abductive logic programming for handling requests between agents, in order to convey some intuition. We have two agents a and b. The beliefs of agent a are represented by the abductive logic program with

P: $get(R,T) \leftarrow \mathbf{request}(a,b,R,T') \wedge \mathbf{accept}(b,a,R,T'') \wedge T'' \geq T' \geq T$
I: $\emptyset$
Ab: $\mathbf{request}(a, Agent, Res, T)$ (action), $\mathbf{accept}(Agent, a, Res, T)$ (observable).

In other words, in order to get a resource, a needs to request it from b and having the request accepted (P). Note that, here and in the sequel, we assume, for simplicity, that accepting a request amounts to getting possession of it.

The goal of a is $get(r, 0)$, for some given resource r. From this goal, the abductive proof procedure generates, by two steps of backward reasoning,

$$\exists T', T''[\mathbf{request}(a, b, r, T') \wedge \mathbf{accept}(b, a, r, T'') \wedge T'' \geq T' \geq 0].$$

The execution model of a selects the abducible $\mathbf{request}(a, b, r, T')$ and executes (utters) it, instantiating T' to, say, 5, and causing b to observe $\mathbf{request}(a, b, r, 5)$. The beliefs of b are given by the abductive logic program with

P: $have(r)$
I: $\mathbf{request}(Y, b, R, T) \wedge have(R) \Rightarrow \exists T'[\mathbf{accept}(b, Y, R, T') \wedge T' \geq T]$
Ab: $\mathbf{accept}(b, Agent, Res, T)$ (action), $\mathbf{request}(Agent, b, Res, T)$ (observable).

In other words, b would grant any agent any requested resource it has.

From the observation $\mathbf{request}(a, b, r, 5)$, the abductive proof procedure generates, by forward reasoning, $have(r) \Rightarrow \exists T'''[\mathbf{accept}(b, a, r, T''') \wedge T''' \geq 5]$, and, by backward reasoning, $true \Rightarrow \exists T'''[\mathbf{accept}(b, a, r, T') \wedge T''' \geq 5]$, which simplifies to $\exists T'''[\mathbf{accept}(b, a, r, T') \wedge T''' \geq 5]$. The execution model of b selects the abducible and executes (utters) it, instantiating T''' to, say, 10, and causing a to observe $\mathbf{accept}(b, a, r, 10)$, instantiating T'' to 10 and thus allowing a to achieve its original goal $get(r, 0)$. The set $E = \{\mathbf{request}(a, b, r, 5), \mathbf{accept}(b, a, r, 10)\}$ is an explanation for both the goal $get(r, 0)$ of a and the observation $\mathbf{request}(a, b, r, 5)$ of b, for the abductive logic programs of the respective agents. As far as a is concerned, both abducible atoms are in E in order to entail the goal. As far as b is concerned, $\mathbf{request}(a, b, r, 5)$ is in E in order to (trivially) entail the observation and $\mathbf{accept}(b, a, r, 10)$ is in E to satisfy the integrity constraint of b.

3 Dealing with Requests of Resources

Until section 5 we will model, via an abductive logic program, the knowledge of some agent a in the system. This abductive logic program contains agent-independent if-rules and if-then-rules, that a shares with the other agents in the system, as well as agent-dependent if-rules, that are specific to a. Until section 5 we will concentrate on the agent-independent part of the the knowledge base of a, that can be adopted by other agents too, simply by replacing a with the name of the agents whenever required.

Figure 1 summarises the knowledge of agent a that a uses to obtain resources. By the if-rules P1-P3, a identifies all resources it needs and goes about getting them all, one by one. Instances of *need* are defined by agent-dependent if-rules, e.g as in section 5. The arguments T_0 and T indicates the time the agent starts and finishes (respectively) its search for the resources. A possible definition of *choose* is given by P15, assuming that sets are represented as lists.

By P4-P5 and P7-P8, a first attempts to get each resource within the societies of which a is a member and in which a believes the resource is available, by

(P1) $\mathit{get\_all\_needed\_resources}(T_0, T) \leftarrow \mathit{need}(Rs) \wedge \mathit{get\_all}(Rs, T_0, T)$

(P2) $\mathit{get\_all}(Rs, T, T) \leftarrow Rs = \emptyset$

(P3) $\mathit{get\_all}(Rs, T_0, T) \leftarrow Rs \neq \emptyset \wedge \mathit{choose}(R, Rs, Rs') \wedge$
$\mathit{get}(R, T_0, T') \wedge \mathit{get\_all}(Rs', T', T)$

(P4) $\mathit{get}(R, T_0, T) \leftarrow \mathit{member\_socs}(a, R, T_0, Ss) \wedge \mathit{get\_from}(R, T_0, T, Ss)$

(P5) $\mathit{get\_from}(R, T_0, T, Ss) \leftarrow Ss \neq \emptyset \wedge \mathit{choose}(S, Ss, Ss') \wedge$
$\mathbf{request}(a, S, R, T') \wedge T' \geq T_0 \wedge$
$\mathit{process}(S, R, T', T, Ss')$

(P6) $\mathit{get\_from}(R, T_0, T, Ss) \leftarrow Ss = \emptyset \wedge \mathit{non\_member\_socs}(a, R, T, Ss') \wedge$
$\mathit{get\_join}(R, T_0, T, Ss')$

(P7) $\mathit{process}(S, R, T_0, T, Ss) \leftarrow \mathbf{accept}(A, a, S, R, T)$

(P8) $\mathit{process}(S, R, T_0, T, Ss) \leftarrow \mathbf{refuse}(A, a, S, R, T') \wedge \mathit{get\_from}(R, T', T, Ss)$

(P9) $\mathit{get\_join}(R, T_0, T, Ss) \leftarrow Ss \neq \emptyset \wedge \mathit{choose}(S, Ss, Ss') \wedge$
$\mathit{try\_to\_join}(S, T_0, T') \wedge \mathit{process\_join}(S, R, T', T, Ss')$

(P10) $\mathit{process\_join}(S, R, T_0, T, Ss) \leftarrow \mathit{joined}(a, S, T') \wedge$
$\mathbf{request}(a, S, R, T'') \wedge T'' \geq T' \wedge$
$\mathbf{accept}(A, a, S, R, T)$

(P11) $\mathit{process\_join}(S, R, T_0, T, Ss) \leftarrow \mathit{joined}(a, S, T') \wedge$
$\mathbf{request}(a, S, R, T'') \wedge T'' \geq T' \wedge$
$\mathbf{refuse}(A, a, S, R, T''') \wedge$
$\mathit{get\_join}(R, T''', T, Ss)$

(P12) $\mathit{process\_join}(S, R, T_0, T, Ss) \leftarrow \mathbf{deny}(A, a, S, T') \wedge \mathit{get\_join}(R, T', T, Ss)$

(P13) $\mathit{member\_socs}(X, R, T, Ss) \leftarrow \mathit{findall}(S, (\mathit{member}(X, S, T) \wedge$
$\mathit{available}(R, S)), Ss)$

(P14) $\mathit{non\_member\_socs}(X, R, T, Ss) \leftarrow \mathit{findall}(S, (\neg \mathit{member}(X, S, T) \wedge$
$\mathit{available}(R, S)), Ss)$

(P15) $\mathit{choose}(X, Y, Z) \leftarrow Y = [X|Z]$

(I1) $\mathit{get\_join}(R, T_0, T, Ss) \wedge Ss = \emptyset \Rightarrow \mathit{false}$

Fig. 1. Obtaining resources: knowledge base of agent a

repeatedly choosing one such society until one of its requests for that resource is accepted. By P6 and P9-P12, if one resource cannot be obtained within the societies to which a belongs, a attempts to join other societies, one at a time, until it succeeds in joining one such society and one of its requests for the required resource is accepted within the newly joined society. By I1, if there are no more societies worth joining for the purposes of getting a resource, as all potentially useful societies have denied a to join or have admitted a but have refused to accept the request for the given resource, then a will be unable to obtain that resource and thus will fail, represented by false.

The predicates *member*, *try_to_join* and *joined* are defined in figure 3 and will be discussed in section 4. Instances of *available* are defined by agent-dependent if-rules, e.g as in section 5.

Figure 2 summarises the knowledge of agent a for processing requests from other agents. This knowledge is only used if a controls some resource in some society. If the request comes from an agent who is a member of the society to

$$\begin{aligned}
&\text{(I1) } \mathbf{request}(X,S,R,T) \wedge \mathit{control}(a,R,S) \wedge \mathit{member}(X,S,T) \wedge \mathit{available}(R,S) \Rightarrow \\
&\qquad\qquad \exists T'[\mathbf{accept}(a,X,S,R,T') \wedge T < T'] \\
&\text{(I2) } \mathbf{request}(X,S,R,T) \wedge \mathit{control}(a,R,S) \wedge \neg \mathit{member}(X,S,T) \Rightarrow \\
&\qquad\qquad \exists T'[\mathbf{refuse}(a,X,R,S,T') \wedge T < T'] \\
&\text{(I3) } \mathbf{request}(X,S,R,T) \wedge \mathit{control}(a,R,S) \wedge \neg \mathit{available}(R,S) \Rightarrow \\
&\qquad\qquad \exists T'[\mathbf{refuse}(a,X,R,S,T') \wedge T < T']
\end{aligned}$$

Fig. 2. Processing requests: knowledge base of agent *a*

which the request is made then, and if the resource is indeed available in that society, then the request is accepted (I1), whereas, if the request is not available in that society, the request is refused (I2). The request is also refused if it comes from an agent who is not a member of the society (I3). Note that *a* would only handle requests for resources it controls. Instances of *control* are defined by agent-dependent if-rules, e.g as in section 5.

Note that, since resources are assumed to be either always available or always unavailable (due to the predicate *available* not being time-stamped), requests by members are always accepted by agents controlling the requested resources, and no concept of the resource being free at the time of the request is needed. Also, resources are considered to be reusable and can be used by multiple users simultaneously.

4 Joining/Leaving Societies and Admitting/Rejecting New Members

Figure 3 summarises the knowledge of agent *a* that *a* uses to join a society, on the basis of its need for resources. Trying to join a society amounts to simply joining if the society is open (P1) and joining after having performed a self-imposed eligibility check, if the society is semi-open or semi-closed (P2 or P3, respectively). Note that the actual eligibility criteria of the society might be different from the ones the agent believes they are. In particular, the society might perform additional checks, e.g. concerning the trustworthiness of the agent, that the agent might not perform itself.

The predicates *open*, *semi_open*, *semi_closed* and *eligible* are defined by agent-dependent if-rules, e.g as in section 5.

An agent can be thought of as having actually joined a society immediately as it joins, if the society is open (P6), or after having been accepted, either by being admitted directly, if the society is semi-open (P7), or by being assigned a representative, if the society is semi-closed (P8). Note that the if-rules P6-P8 can be used by *a* to reason about itself, with $X = a$, to determine whether *a* has actually joined a society, or about some agent different from *a*, to determine whether such agent is member of some society when, e.g., this agent has put forward a request and a decision needs to be made by *a* on whether to accept the resource or not (see section 3).

(P1) $try\_to\_join(S, T1, T2) \leftarrow open(S) \wedge \mathbf{join}(a, S, T2) \wedge T1 < T2$
(P2) $try\_to\_join(S, T1, T2) \leftarrow semi\_open(S) \wedge eligible(a, S, T1) \wedge$
$\mathbf{join}(a, S, T2) \wedge T1 < T2$
(P3) $try\_to\_join(S, T1, T2) \leftarrow semi\_closed(S) \wedge eligible(a, S, T1) \wedge$
$\mathbf{join}(a, S, T2) \wedge T1 < T2$
(P4) $member(X, S, T) \leftarrow member(X, S, 0) \wedge 0 < T \wedge \neg left(X, S, 0, T)$
(P5) $member(X, S, T) \leftarrow joined(X, S, T') \wedge T' < T \wedge \neg left(X, S, T', T)$
(P6) $joined(X, S, T) \leftarrow open(S) \wedge \mathbf{join}(X, S, T)$
(P7) $joined(X, S, T) \leftarrow semi\_open(S) \wedge \mathbf{admit}(Y, X, S, T)$
(P8) $joined(X, S, T) \leftarrow semi\_closed(S) \wedge \mathbf{represent}(Y, X, A, S, T)$
(P9) $joined(X, S, T) \leftarrow \mathbf{broadcast\_joined}(Y, S, X, T)$

Fig. 3. Joining societies: knowledge base of agent a

An agent can be also thought as having joined a society, and thus being one of its members, from the moment that this information has been broadcast (P9).

If an agent (either a or not) has actually joined a society, then, by the if-rule P5, the agent is a member of that society until it leaves it. Moreover, an agent is to be thought of as a member of a society if it was a member of it at the initial time 0 and until it has left it. The predicate $left$ is defined in figure 4.

Figure 4 summarises the knowledge of agent a relative to leaving societies. By I1, a decides to leave a society when it realises that being in that society is of no benefit, since it does not need any of the resources that are available in that society or, if it does need some resource, this has been refused to it (P1). Note that, by I1, a is not allowed to leave the society if either it is the authority in the a society or it controls some of the resources in that society. Here $authority(a, S)$ represents the fact that the agent a, whose knowledge we are defining, has authority in S to accept or reject new applicants.

The remaining if-then-rules in figure 4 only apply to a if a has the authority over some society. By I2, a will expel any un-trustworthy agent in a society over which it has authority. By I4, a will broadcast to all agents in that society that the given agent has been expelled. By I3, a will broadcast to all agents in a society over which it has authority that some agent has left.

The predicates $authority$, $control$ and $untrustworthy$ are defined by agent-dependent if-rules, e.g as in section 5.

The if-rules P3-P5 define the predicate $left$: an agent X (either a or another agent) is to be considered as having left a society if it has voluntarily left it (P3), if it has been expelled from it (P4), or if there has been a broadcast message within the society that the agent X has left.

Figure 5 summarises the knowledge of a relative to dealing with memberships applications. The if-then rules in this figure all apply only if a has the authority over some society. The if-then-rules I1-I2 express the conditions under which a new applicant is accepted, and deal with the case of semi-open (I1) and semi-closed societies (I2). In both case, a needs to perform an eligibility check on the applicant and broadcast to all agents in the society that a new member has been

(I1) $member(a, S, T) \wedge no\_benefit(S, T) \wedge \neg authority(a, S) \wedge control\_nothing(a, S) \Rightarrow \exists T'[\mathbf{leave}(a, S, T') \wedge T < T']$

(I2) $untrustworthy(X, T) \wedge member(X, S, T) \wedge authority(a, S) \Rightarrow \exists T'[\mathbf{expell}(a, X, S, T') \wedge T < T']$

(I3) $\mathbf{leave}(X, S, T) \wedge member(X, S, T) \wedge authority(a, S) \Rightarrow \exists T'[\mathbf{broadcast\_left}(a, S, X, T')]$

(I4) $\mathbf{expell}(a, X, S, T) \Rightarrow \exists T'[\mathbf{broadcast\_left}(a, S, X, T')]$

(P1) $no\_benefit(S, T) \leftarrow findall(R, available(R, S), Z) \wedge forall(in(R, Z), no\_benefit\_resource(R, T))$

(P2) $control\_nothing(X, S) \leftarrow \neg \exists R[control(X, R, S)]$

(P3) $left(X, S, T1, T2) \leftarrow \mathbf{leave}(X, S, T) \wedge T1 < T \leq T2$

(P4) $left(X, S, T1, T2) \leftarrow \mathbf{expel}(A, X, S, T) \wedge T1 < T \leq T2$

(P5) $left(X, S, T1, T2) \leftarrow \mathbf{broadcast\_left}(Y, S, X, T) \wedge T1 < T \leq T2$

(P6) $no\_benefit\_resource(R, T) \leftarrow \neg need\_resource(R)$

(P7) $no\_benefit\_resource(R, T) \leftarrow \mathbf{accept}(X, a, R, S, T') \wedge T' < T$

(P8) $no\_benefit\_resource(R, T) \leftarrow \mathbf{refuse}(X, a, R, S, T') \wedge T' < T$

(P9) $need\_resource(R) \leftarrow need(Rs) \wedge R \in Rs$

Fig. 4. Leaving societies: knowledge base of agent a

accepted. In the case of a semi-open society, the applicant is notified of admission, via the utterance in the **admit** predicate. In the case of semi-closed societies, a first needs to choose a proxy for the applicant, notify the proxy if its new task of representing the applicant within the society, and then notify the applicant that it has been accepted and that it is going to be represented by the chosen proxy. The atom $proxy(A, S, T)$ holds for those agents A in the society S that can act on behalf of other agents outside the society. The predicate *proxy* is defined by agent-dependent if-rules, e.g. as in section 5. The if-then rule I5 deals with the case of open societies, where any applicant is accepted automatically, without notification, and all a needs to do is to broadcast about the new member. The if-then-rules I3-I4 express that non-eligible applicants are denied membership, both for semi-open (I3) and semi-closed societies (I4).

5 Example

Consider the "travelling businessman scenario" outlined in the introduction, and the situation where, while the businessman is waiting at the airport gate to catch his plane, his agent a, knowing that the businessman will need information about trains at the time of arrival, registers the (device owned by) the businessman with a service providing up-to-date train information in the global computing infrastructure of the country that the businessman will be visiting.

We describe, step by step, the reasoning and behaviour of agent a, assuming that a holds the knowledge given in sections 3 and 4 as well as the following a-dependent if-rules

$member(a, s_1, 0)$	(a is a member of the mobile phone network s_1)
$need(a, \{r\})$	(a needs the train timetable r for some destination)
$available(r, s_2)$	(r is available from the service provider for train info s_2)
$semi\_open(s_2)$	(s_2 is a semi-open society)
$eligible(X, s_2, T)$	(a believes that anybody is eligible to s_2)
$member(d, s_2, 0)$	(a believes that d is a member of s_2)

The resource r is indeed available in the society s_2, which contains two members b and c (and possibly other members), of which b has authority over the society and d controls r. We assume that agents b, c hold the knowledge given in sections 3 and 4 for agent a, but with the symbol a replaced by the symbols b, c, respectively, whenever needed, as well as the following b, c-dependent if-rules:

b: $member(b, s_2, 0)$
$authority(b, s_2)$
$eligible(X, s_2, T) \leftarrow member(X, s_1, T)$
$untrustworthy(d, T)$
$semi\_open(s_2)$

c: $member(c, s_2, 0)$
$control(c, r, s_2)$
$eligible(X, s_2, T) \leftarrow \neg untrustworthy(X, T)$
$available(r, s_2)$

Note that the agents have a partial knowledge of members of societies, e.g. b, c do not know/believe that a is a member of s_1, and neither b nor c knows/believes that the other is a member of s_2. Also, the agents may have inaccurate knowledge of societies, e.g. a believes that d is a member of s_2, which is not the case (because if it were the case, b, the authority in s_2, would know it). Moreover, the agents might hold conflicting beliefs, e.g. b and c have different beliefs about the eligibility criteria for s_2 (the only one that matters is the one held by the authority b). Further, the information about the type of societies and the availability of resources in societies is common to all relevant agents. Thus, this information can be thought of as global, rather than local to individual agents. In general, such information could be local, and possibly partial and inaccurate. Finally, for simplicity we assume that in each society there is only one agent with authority to deal with applicants and only one controller for each resource. The authority does not need to know who controls which resource in a society.

(I1) $\mathbf{join}(X, S, T) \land authority(a, S) \land semi\_open(S) \land eligible(X, S, T) \Rightarrow$
$\exists T1, T2[\mathbf{broadcast\_joined}(a, S, X, T1) \land \mathbf{admit}(a, X, S, T2) \land T < T1 < T2]$
(I2) $\mathbf{join}(X, S, T) \land authority(a, S) \land semi\_closed(S) \land eligible(X, S, T) \Rightarrow$
$\exists Y, T1, T2, T3[proxy(Y, S, T) \land \mathbf{broadcast\_joined}(a, S, X, T1) \land$
$\mathbf{assign}(a, Y, S, X, T2) \land \mathbf{represent}(a, X, S, Y, T3) \land \land T < T1 < T2 < T3]$
(I3) $\mathbf{join}(X, S, T) \land authority(a, S) \land semi\_open(S) \land \neg eligible(X, S, T) \Rightarrow$
$\exists T1, T2[\mathbf{deny}(a, X, S, T1) \land T < T1 < T2]$
(I4) $\mathbf{join}(X, S, T) \land authority(a, S) \land semi\_closed(S) \land \neg eligible(X, S, T) \Rightarrow$
$\exists T1, T2[\mathbf{deny}(a, X, S, T1) \land T < T1 < T2]$
(I5) $\mathbf{join}(X, S, T) \land authority(a, S) \land open(S) \Rightarrow$
$\exists T1[\mathbf{broadcast\_joined}(a, S, X, T1) \land T < T1]$

Fig. 5. Admitting/rejecting new members: knowledge base of agent a

Let us now describe the behaviour of a, as given by executing an abductive proof procedure for a's goal $get\_all\_needed\_resources(0, T)$. First we obtain

$get(r, 0, T)$	(by repeatedly unfolding with P1-P3 in figure 3)
$get\_from(r, 0, T, \emptyset)$	(by unfolding with P4 and P13 in figure 3)
$get\_join(r, 0, T, \{s_2\})$	(by unfolding with P6 and P14 in figure 3)

Then, by unfolding with P9-P10 in figure 1, we obtain:

$$\exists T', T'', T'''[try\_to\_join(s_2, 0, T') \wedge joined(a, s_2, T'') \wedge \\ \mathbf{request}(a, S, r, T''') \wedge \mathbf{accept}(X, a, S, R, T) \wedge T''' > T'']$$

Now the atom in the predicate try_to_join can be unfolded, using the if-rules in figure 3, thus giving, since s_2 is semi-open and since a passes its own eligibility check for s_2

$$\exists T', T'', T'''[\mathbf{join}(a, s_2, T') \wedge joined(a, s_2, T'') \wedge \\ \mathbf{request}(a, s_2, r, T''') \wedge \mathbf{accept}(X, a, S, R, T) \wedge T''' > T'']$$

At this point, the execution model selects the communicative action (abducible) in the predicate **join** and executes it, instantiating its time parameter, e.g. to $T' = 10$. The execution of this action by agent a corresponds to the observation by all members of s_2 of $\mathbf{join}(a, s_2, 10)$. This observation will trigger the authority agent b in s_2 to propagate with the if-then-rule I1 in figure 5 and to execute actions $\mathbf{broadcast\_joined}(b, s_2, a, T2)$, with $10 < T2$, and $\mathbf{admit}(b, a, s_2, T3)$, with $T2 < T3$. Assume the execution models of b instantiates $T3$ to 20. Agent a will observe $\mathbf{admit}(b, a, s_2, 20)$, giving, after unfolding $joined$ with P7 in figure 3,

$$\exists T'''[\mathbf{join}(a, s_2, 10) \wedge \mathbf{admit}(b, a, s_2, 20) \wedge \\ \mathbf{request}(a, S, r, T''') \wedge \mathbf{accept}(X, a, S, R, T) \wedge T''' > 20]$$

Similarly as before, the execution model selects the communicative action in the predicate **request** and executes it, instantiating its time parameter, e.g. to $T''' = 30$. The execution of the **request** action by agent a corresponds to the observation by all members of s_2 of $\mathbf{request}(a, s_2, r, 30)$. Repeating a similar process as the one outlined before, agent c grants the resource r to a, by triggering I1 in figure 2, since agent c controls r in s_2. After having obtained the resource, a will leave s_2, by reasoning as before but with I1 in figure 4, since there is no benefit anymore for a to stay in s_2.

6 Concluding Remarks

We have applied abductive logic programming to specify the knowledge and behaviour of software agents that need to access resources in a global computing environment populated by artificial societies. Our investigation has focused on the situation where an agent that needs a resource has to join artificial societies that control the provision of that resource.

We have made a number of simplifying assumptions. We have not taken into account applications for temporary membership in a society. We have assumed that the definitions of *authority*, *control*, *open*, *semi_open*, *semi_closed* and *available* do not change over time. We have assumed that resources are needed until they have been obtained, and we have ignored the possibility that resources may be needed all the time, or for specific periods. Also, we have not allowed the generation of new resources within societies, the creation of new societies, the creation of new agents in societies, and the assignment of new roles (authority/control) to agents in societies. As a consequence, we disallow agents holding authority and control to leave societies. Furthermore, we give no justification of refusal of resources, denial of admittance in a society, or reasons for an agent to leave while broadcasting. Future work is needed to extend the framework to relinquish these assumptions. Finally, we have assumed that, whenever agents need to share knowledge, they do so through the explicit replication of this knowledge in their knowledge bases (abductive logic programs). As a result, when new members join or leave a society, the authorised agent need to notify all agents in the society. This of course may cause communication bottlenecks. In order to cut-down communication we could extend the framework to access – through observations – the objects available in the shared state managed by the environment (for example as in [17]). Such an object could be a *yellow pages* resource, maintaining information about available resources within different societies.

However, note that the in our approach agents do not need to share all their knowledge. In particular, agents might adopt different eligibility criteria. Indeed, the communication-based approach that we have taken allows encapsulation of the knowledge of agents and guarantees the privacy of agents in societies.

Communication in our framework does not rely on mentalistic semantics, such as the one envisaged by FIPA. Instead, we understand communication purely by means of abduction.

In our approach, artificial societies are not merely locations providing resources. Indeed, societies are characterised by a set of conventions, e.g. in deciding the admission of new members, which are held by the authority within a society, or in deciding when a request should be granted, which are held by the controller of that resource. Note that, for simplicity, we have assumed that all societies share the same such conventions, but different ones could be easily incorporated within the approach. For example, a controller could grant some resources only to members that have newly joined, or to members that have been part of the society for a specified period.

At a very general level, our work is influenced by the research programme outlined in [10], whereby artificial societies support and augment the activities of people in human societies. We share the idea of [10] of having electronic agents acting on behalf of people, and that these societies should be situated in the physical environment of real ones. However, we differ from that work in that we do not rely upon the *ownership* model that it advocates. Instead, we focus on the notion of *service*, viewing a service as a resource that is available in specific societies. Our approach is neutral as to which model of ownership is being adopted.

Our work also closely relates to [1] in that we use the classification scheme for artificial societies that it proposes. We have investigated how this classification scheme can be utilised by agents to reason about joining artificial societies, for the purposes of accessing resources in a flexible manner. We have also provided a computational framework that demonstrates the usefulness of this classification, by instantiating it with respect to a specific representation for agents.

Our computational approach extends existing work on the application of abductive logic programming to the resource re-allocation problem [12, 13, 14]. There, the same abductive logic programming approach that we have used is used to show how agents can negotiate the exchange of resources in a multi-agent system environment. Our work complements that approach by providing a framework where the accessing of resources is dependent on globally situated societies that control the resources. Future work is needed to provide a formal analysis, along the lines of the formal analysis in [13, 14], of the framework we have proposed. Some interesting properties to be analysed would be: *"An agent only belongs to a society if this is beneficial to the agent or the agent has responsibilities within that society"*, *"If a resource is available in some society and the agent is eligible to be a member of that society, the agent will obtain the resource in finite time"*.

Acknowledgements

This research was supported by the EU-funded project SOCS, [15]. The authors would like to thank the anonymous referees for helpful comments.

References

[1] P. Davidsson, Categories of artificial societies, [9]. 127, 139

[2] P. Dell'Acqua and L. M. Pereira, Preferring and updating in abductive multi-agent systems. [9]. 128

[3] P. Dell'Acqua, F. Sadri, F. Toni, Combining Introspection and Communication with Rationality and Reactivity in Agents, *Proc. JELIA'98*, U. Furbach, L. Farinas del Cerro eds., Springer Verlag LNAI 1489, 17-32, 1998. 128

[4] T. H. Fung, R. A. Kowalski, The iff procedure for abductive logic programming. *Journal of Logic Programming* 33(2):151–165, Elsevier. 130

[5] Global Computing, Future and Emerging Technologies Web-site, http://www.cordis.lu/ist/fetgc.htm (visited 10/06/2002). 126

[6] A. C. Kakas, R. A. Kowalski, F. Toni, The role of abduction in logic programming. *Handbook of Logic in AI and Logic Programming* 5:235–324, OUP, 1998. 128, 130

[7] R. A. Kowalski, F. Sadri, From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, Baltzer Science Publishers, J. Dix, J. Lobo (Eds), volume 25(3,4):391–420, 1999. 128, 130

[8] A. Mamdani, J. Pitt, K. Stathis, Connected communities from the standpoint of multi-agent systems. *New Generation Computing Journal*, Special issue on New Challenges in Intelligent Systems, 17(4), T. Nishida (ed.), 1999. 127

[9] A. Omicini, P. Petta, R. Tolksdorf (Eds.): *Proc. ESAW'01*, Springer Verlag LNAI 2203, 2001. 127, 139

[10] J. V. Pitt, A. Mamdani, and P. Charlton, The open agent society and its enemies: a position statement and research programme, [18], pp 67-87. 138

[11] F. Sadri, F. Toni, Abduction with negation as failure for active and reactive rules. *Proc. AI*IA 99*, E. Lamma, P. Mello eds, Springer Verlag LNAI 1792, 49–60, 2000. 128, 130

[12] F. Sadri, F. Toni, P. Torroni, Logic Agents, Dialogue, Negotiation - an Abductive Approach, *Proc. AISB*, M. Schroeder, K. Stathis eds, 2001. 128, 139

[13] F. Sadri, F. Toni, P. Torroni, Dialogues for negotiation: agent varieties and dialogue sequences, *Proc. ATAL'01*, J. J. Maher ed., LNAI 2333, 405–421, 2001. 128, 139

[14] F. Sadri, F. Toni, P. Torroni, Resource reallocation via negotiation through abductive logic programming, Proc. *JELIA 2002*, to appear. 128, 130, 139

[15] SOCS (Societies Of ComputeeS) Technical Annex, IST-2001-32530, 2001, `http://lia.deis.unibo.it/Research/Projects/SOCS/`. 127, 139

[16] K. Stathis, *Location-aware SOCS: The 'Leaving San Vincenzo' Scenario*, SOCS Technical Report, IST32530/CITY/002/IN/PP/a2, 2002. 127

[17] K. Stathis, *A game-based architecture for developing interactive components in computational logic*, J. of Functional and Logic Programming, 2000 (1), MIT Press. 138

[18] K. Stathis and P. Purcell (Eds.): Special issue on *LocalNets: Environments for Community-based Interactive Systems*, Journal of Telematics and Informatics, Elsevier, 18(1), 2001. 140

[19] K. Stathis, O. deBruijn, and S. Macedo, Living memory: agent-based information managment for connected local communities, *Interacting with Computers*, to appear. (Also available in: http://dx.doi.org/10.1016/S0953-5438(02)00014-0). 127

[20] F. Toni, Automated Information Management via Abductive Logic Agents ,[18], pp 89-104. 128

Motivating Participation in Peer to Peer Communities

Julita Vassileva

Computer Science Department, University of Saskatchewan
57 Campus Drive, Saskatoon, Saskatchewan, S7N 5A9 Canada
jiv@cs.usask.ca

Abstract. One of the most important prerequisites for the success of a peer to peer system is the availability of participants willing to provide resources (files, computational cycles, time and effort to the community). Free riders may do no harm in file-sharing applications, like NAPSTER, because of the nature of electronic data, which can be reproduced at no cost; downloading a copy does not take anything away from the common resources. However, free riders can be destructive in applications where there are costs associated with the resources shared. The paper argues that providing motivation or some kind of incentives for users to participate is very important. It describes various methods to motivate different kinds of users and describes a design of a peer to peer system called Comutella, which is being developed currently to support file and service (help, advice) sharing in research groups and groups of learners.

1 Introduction

Peer to peer (P2P) systems have become increasingly popular in recent years. Applications like Napster, KaZaA the myriad of Gnutella-based file-sharing software or SETI@home have given publicity of P2P computing. P2P systems are considered "invincible" because of the decentralization of control (e.g. Gnutella), "unbeatable" in terms of performance (e.g. Seti@home), and moreover, enthusiastically supported by all the participating users. One of the most important prerequisites for the success of a P2P system is the availability of participants willing to provide resources (files, computational cycles, time and effort) to the community. Studies [2] have shown that P2P systems are prone to be overwhelmed by "free riders", i.e. people who do not contribute resources, but only consume. While this has given rise to some pessimistic expectations about the future of P2P computing, others [14] have argued that free riders are not be harmful in file-sharing P2P applications, like NAPSTER or KaZaA, because of the nature of electronic data. It can be reproduced at no cost; downloading a copy does not take anything away from the common resources, but instead creates a new resource that can be shared. However, the problem remains in P2P applications where there are costs associated with the resources shared, e.g. compute cycles, network bandwidth, or human time and effort. In such systems, free riders can be a

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 141-155, 2003.

menace, and they can bring down the functionality for other users (and increasingly, for themselves) by their consumption of resources, without contributing anything.

A P2P application, called COMUTELLA (Community Gnutella) is currently being implemented at the Mobile and Ubiquitous Computing Lab at the University of Saskatchewan. The system will enable research or study groups of students to collaborate and share resources, e.g. to exchange both files (e.g. research papers) and services (e.g. help each other). Each user installs a servent (a small application which is both a client and a server) that allows the user to search for resource or services and to offer resources and services to community. The servents communicate with each other using the Gnutella protocol. The nature of resources that are shared requires active involvement of the users (e.g. finding relevant research papers, saving and organizing them, enabling folder sharing, or engaging in conversation with other user(s) to answer questions, provide advice etc.). Therefore, it is important to ensure user participation and a certain level of contribution to the community, so that a reliable service is provided.

This paper does not report on a piece of finished work; it rather describes the design of a system that is being developed right now. The remainder of the paper is organized as follows: section 2 discusses various methods of motivating users to participate actively in a P2P community, section 3 presents the user modelling and adaptation that needs to be done by the servent, section 4 discusses methods for motivating users to participate and section 5 concludes the paper.

2 Motivating Users to Participate

Motivation of users to participate in the community is a crucial factor for the success for a P2P system. Our experience with I-Help, a P2P system for help, that has been used for over 2 years in the University of Saskatchewan showed that if the deployment of the system lacks a "critical mass" of active users, it will never be able to take off [5]. Indeed, if each user logs in, asks for help and after finding (or more likely not finding it) logs out, there would be very few people simultaneously on-line. That means that the likelihood of finding a helper, competent about a given question, would be very low. After trying unsuccessfully to find help a couple of times, people stop trying. In this way the pool of users shrinks quickly in a downwards spiral and the system becomes disfunctional. On the contrary, if users stay logged in, there is a higher probability that when somebody asks a question there would be somebody on-line who can answer. A user who received help is more likely to perceive the system as useful and remain logged in (in case she needs it again), available, when her expertise is required. In this way, the amount of participating users, as well as the value of the system increases in a snowball effect.

It is important that the users perceive the system as useful and are willing to participate. We believe that this is true for file sharing P2P systems as well as for systems allowing sharing services or computing tasks. The size of the community of users defines the level of usefulness, or the value of the system, and the other way around, the value of the system defines the number of users. This "feedback loop" can develop by itself, as happened in file-sharing systems like Napster and KaZaA. However, in other applications, like our peer-help system I-Help, it doesn't just happen. Our experience showed, that incentives are needed for the users in the beginning. This can be achieved either by providing an ample amount of resources in the system, or some other incentives. Initial investment of resources, for example, hiring a knowledgeable person, teaching assistant or lecturer, to be constantly on-line and to answer immediately any question, is very helpful. It creates the impression in the students that the system is useful and they get used to asking their questions there, instead of phoning somebody. Once they get into the habit of staying on line, the knowledgeable person can withdraw; the students start getting matched with each other and helping each other. An alternative is to provide an incentive by giving some participation marks for using the system. Even though this reward is external to the system, it can draw a significant amount of users and ensure the "critical mass" necessary for the system to function.

Obviously, these two ways of motivating participation in the P2P system depend on the nature of the application and the context of use (in this case formal University setting). Generally, motivation strategies seem to depend on how much participation or effort is desirable from the users and the value that the system provides to them.
There are several levels of user cooperative participation in a P2P system. They are characterized with decreasing degree of active user involvement (activeness):

- *create service*: creating new resources or services and offering them to the community,
- *allow service*: providing to the community disk space to store files for downloads or computing resources to enable a service that has been created by another participant in the community,
- *facilitate search*: providing a list of relationships to other users to facilitate their search of files or services. This level of cooperativeness is possible if the servents model the "good" relationships with peers, as we propose in the next section,
- *allow communication*: forwarding ping-pong, query and hit messages, i.e. actively participating in the protocol of the network, thus providing the "glue" that holds the network together [2] thus facilitating the peer infrastructure.
- *uncooperative free-rider*: downloading files or utilizing services when needed, but going off line immediately afterwards.

The "create service" level usually includes "allow service", "facilitate search" and "allow communication", i.e. it describes the most socially cooperative type of user behaviour.

The more typical level is "allow service", describing a user who contributes passively to the community, by providing her resources and relationships, as well as the functionality of her servent to enhance the infrastructure of the community, but does not actively bring new resources or services into the system. As shown in [2], in Gnutella only a tiny minority falls into this category - 5% of the users is responsible for sharing over 70% of the files.

According to [2], the majority of users (66%) fall into the category, "allow communication" - they participate in the network infrastructure and therefore can be detected and taken into account. Unfortunately, there is no way to know at any moment how many users are "uncooperative free riders" or "creators of service" due to the lack of history in Gnutella and the anonymity, which does not allow to identify who first introduced a file into the system.

According to [11], more than 80% of the users of Mojo Nation were "1-time, 1-hour" users, and of the remaining users a significant part were "1 time, less than 24 hour" users. We observed a similar behaviour of users in our I-Help system in certain classes, where the instructors failed to motivate a "critical mass" of active users in the beginning, when most of the users log-in just to try the system [5].

All popular file sharing P2P systems, like NAPSTER, Morpheus /KaZaA try to ensure at least the "allow service" and "allow communication" levels of cooperation. Usually "allow service" is ensured by a default setting in the servent, which commands the downloaded files to be saved in the "shared folder", so that they can be found by other servents. The "allow communication" level is achieved by making it harder to quit the servent (even if clicking on the close window button, which normally quits an application, a KaZaA servent will remain active until it is quit once again explicitly by selecting the "Quit" option on the right-click menu). These settings can be changed by the user, e.g. the downloaded files can be saved in a file different from the "Shared Folder", but it requires more knowledge and active involvement from the user, which is a kind of punishment for uncooperative users who wish to be only free riders and not to contribute to the community in any way.

There are four principle ways of motivating users to participate in a community:

- by trying to influence the user's feelings (of guilt, of belonging or owing to the group) to stimulate her altruism towards the community,
- by rewarding the user with visibility / reputation in the group depending on hisher contribution,
- by allowing the user to develop relationships with other users in the community (one would do a favor to a friend, which one might not want to do for anonymous unknown people), and
- by providing an economical model which ensures incentive for user contribution, (e.g. better quality of service, priority in the queues).

It is likely that choosing the appropriate way of motivation depends:

- on the personality of the user, and
- on the nature or the user's interest in the area and the group (community).

Thus, the same user can be altruistic in one group, motivated by reputation in another group and by economic rewards in a third group.

The conclusion is that one needs to know a lot about the user to be able to persuade or motivate her to participate, about the user's interest in the community, about people with similar interests, who might become potential "friends" of the user, and finally about the user herself, in term of how selfish or altruistic she is. User modeling provides means to capture such features of users.

3 User Modelling

To create user groups based on interest, the servant needs to understand the character and the interests of the user and facilitate finding and maintaining relationships with users with similar interests. The approach described below is tied to the COMUTELLA application area - exchanging academic papers and on-line help/discussion on various topics. However, it can be adapted for other similar areas, where users are looking for resources and services that can be classified according to their semantics.

3.1 User Model Representation

The user model contains four different parts:

- a model of the userís personality
- a model of the user's interests,
- a model of the user's relationships,

The model of user personality can be simply a number denoting the level of the user's selfishness, varying from -1 (selfish) to +1 (altruistic).

The model of user interests is represented as a list of topics / areas in which the user is interested. An ontology representing typical search terms in a given semantic area allows clustering users into groups sharing similar interests. It is important to note that these user groups can overlap on various levels, e.g. one user can be a member of a group interested in Bulgarian folk music and in a group interested in blues. The same user can be also a member of a group interested in P2P computing and in a group interested in multi-agent systems. However, she may be a member of a sub-group of the multi-agent systems group interested in agent negotiation and coalition formation and not be a member of a group interested in animated avatar agents (see Fig. 1).

The model of user interests is organized hierarchically as an overlay over the domain ontology. Sub-areas in which the user has shown interest by issuing queries are represented, along with a value that indicates the strength of user interest in the area and a time stamp showing when the user made the last query in this area. The user's strength of interest in an area is calculated based on how many times the user has searched in this area, and how recently she has been searching in this area. The user's interest in areas that are more general (higher in the ontology heterarchy) than the current area of search are also impacted, but in a much weaker way.

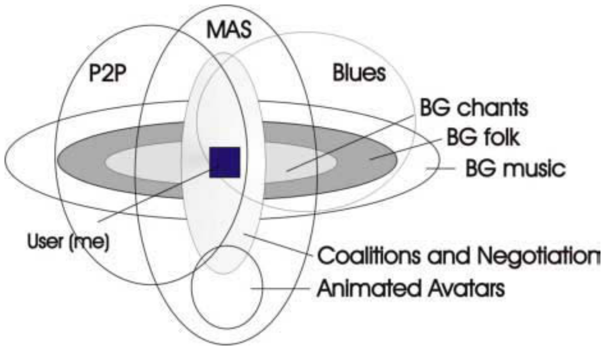


Fig. 1. User interest groups

It is clear that in order to apply this approach, a necessary condition is that all servents use a compatible representation of a domain ontology or ontology of services. There are various tools to developing such ontologies, e.g. DAML-S [3].

The model of the user's relationships includes the users with whom the user has interacted frequently, i.e. from whom the user has downloaded files frequently and also users, who have downloaded files frequently from the user. These relationships are represented in a list where each relationship contains the unique ID of the other user, the search area in the context of which the users have interacted and two numbers representing the strength and the balance of the relationship. The strength is a subjective factor reflecting how satisfied the user was with the interaction, i.e. if she kept the downloaded file or deleted it, if she used the file frequently. The balance of the relationship denotes the direction of services / files, i.e. who of the users predominantly uses and who offers resources / services. The next section explains how these three representations are created and updated.

3.2 Creating and Updating the User Model

A simple method - reinforcement learning - is selected as a basis for maintaining all parts of the user model.

3.2.1 Modelling User Personality

The level of user's selfishness/altruism is updated based on user actions that have impact on the community, e.g. providing new files to be shared, enabling file-sharing for folders on the user's hard disk, the preferences the user has chosen with respect to file-transfers from other people (see Fig.3). For example, if the user wants to bring a new resource (paper) in the system, she has to spend some effort to annotate the file with respect to category, so that it is searchable. If the user is willing to do this, the level of altruism of the user is increased. If the user interrupts a on-going transfer, this is a sign of a selfish behaviour, and the level of altruism is decreased. Other parameters considered in the computation of the personality characteristic of the user are the number of files shared by the user, the relative duration in which the user's servent is active, and the number of user actions that are deemed as uncooperative, such as removing downloaded files from the shared folder, or disallowing sharing of folders.

3.2.2 Modelling User Interests

Each servent keeps track of the areas entered by the user for search in the model of user interests. The areas reflect the ontology of the domain. In COMTELLA, these are a subset of the ACM set of subject categories. The strength of user interest S^a at time *t* in each sub-area *a* in the ontological heterarchy that is on the path leading to the sub-area related to the words of query is updated according to the reinforcement learning formula:

$$S^a(e_t, t) = i * S^a(e_{t-1}, t-1) + (1 - i) * e_t \qquad (1)$$

where the new (at time *t*) evidence of interest $e_t \in [0, 1]$ is calculated as $e_t = 1/ d$, where $d = 1 +$ *the distance* between the level of the current search area in the ontology tree and the level of the area *a*.

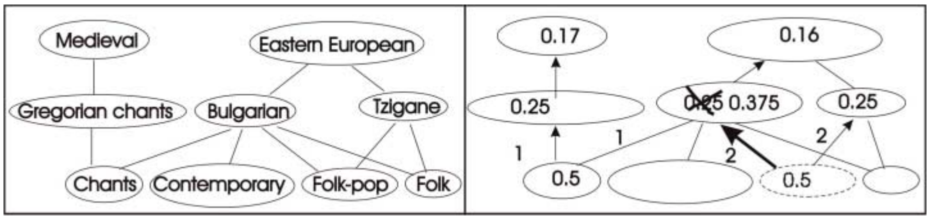


Fig. 2. Updating the model of user interests. The numbers associated with arcs on the right indicate the order of updating; first evidence about user interest in chants is propagated; next evidence about user interest in folk-pop is propagated

The parameter $i \in [0.5, 1]$ is an inflation rate used to model the fact that older experiences become less important over time *t*, while the most recent experience is the most relevant (since the user's preferences may change over time). It can be fixed at a given value, say 0.5, giving equal weights to old and new evidence. The parameter *i* can also be a variable, which depends on the time elapsed since the last evidence of interest in this area, which allows capturing better the current tendency in user interests. An example is shown in Figure 2.

1.1.3 Modelling User Relationships

To model the relationships of the user, the servent keeps track of the following:

- which servents respond with relatively many hits to the user's query,
- from which servents the user chooses to download files or to request service,
- the success of each download / service,
- who issues queries that result in hits in the list of resources of the user,
- who downloads files from the user.

The servents who have returned many hits along with those from whom the user has chosen to request service or download a file are entered in the user's list of "relationships" with an indication of the context of the particular area of interest (request) in which the relationship was created. The success of each download or service is used to update the strength of the relationship between the users using a formula similar to (1). Servents who are searching for files / services that are offered by the user and those who choose to download files or use the services offered by the user are added to the list of "relationships" of the user in the context of the particular area of interest depending on the query used for the search.

In addition to the relationship's strength and context, the servent keeps track of the balance (reciprocity) of the relationship. The servent of user X calculates the balance of its relationship with the servant of user Y as:

$$B^{XY} = N^{X\text{✐}Y} - N^{Y\text{✐}X} \tag{2}$$

i.e. the difference between the number of times when the user X has downloaded files from Y and the number of times when user Y has downloaded files from X. If the balance is negative, the user X "owes" user Y.

The sum of the balances of all relationships of a user defines how much she has contributed to the community and how much she has consumed. Keeping a balance of each relationship allows maintaining a model of the user's contribution to individual users, to every interest group in which she participates and to the network as a whole. It is used, along with the model of the user's personality by the servent to provide an individualized motivational interface for the user, to persuade her/him to contribute to the community.

1.1.4 Sharing User Models Among Servents

While not involved actively in search, servents can communicate with each other and learn about other users with the similar interests. This learning can take two forms:

- *Direct:* without explicit request from the user, the servent sends queries in a particular area to find out servents that have resources and enters them in the list of relationships of the user. The strength of relationship can be minimal in this case, calculated as in [8] by the percentage of the number of hits by this servant over the total number of hits. In this way the servent explores the network. Also the time of request and reply can be recorded, to capture the compatibility in time patterns of being on-line.
- *Indirect:* by requesting the list of their relationships in a given area from other servents, with whom the servent has strong relationships in this area. In this way the IDs of servents that have provided good files in a given area, have been frequently available and have provided a good service (e.g. did not interrupt the connection during download) are shared among the servents.

Through this collective learning, in a network which is not very dynamic, i.e. most of the servents are active at approximately the same time, ultimately, all communicating agents from one interest group would end up with the same list of relationships, which will lead to an implicit objective measure of quality / ranking (or reputation) of each servent within the group. This objective ranking will be contextualized, i.e. it will make sense only in the context of one interest group, since people behave differently in different communities. Of course, this will not prevent servents who have a high rank in one group to have a high rank in another group too.

There are a number of possible strategies concerning the requesting and interpretation of information received from others. For example, we assume that a servent will ask only the servents with the highest rank in its list of relationships about their relationships. However, it is possible to request data from all existing "acquaintances" and use the strength of relationship with each source to compute the strength of relationship in the new "acquaintance". This approach is similar to approaches for trust propagation among agents in multi-agent communities [13].

There are many open questions, for example, how information about a given servent coming along a chain of "acquaintances" should be interpreted. There are various possibilities: by averaging the strength values along the path, or by multiplying them. It is also necessary to define a policy for resolving conflicts between different two different chains of sources. Should two separate representations of the strength of relationship be kept by the servent: a subjective one based on the servent's own experience, and objective based on information received from other servents. How can these two representations be combined and when?

There are arguments against using global reputation measures. For example, if user X requests a service from user Y always at 1 a.m. and never gets anything from him, since Y is never on-line at that time, X's strength of relationship with Y would be 0. However, Y could be a very active member of the community, providing useful files and services at other times. The strength of relationship that X has with Y reflects not just the common interests, but also the compatibility between X's and Y's preferences in the time pattern of usage, and it cannot be generalized without attaching a lot of context information. Just averaging the strength of relationship values of many people without considering the contextual information would not be appropriate. More sophisticated techniques are necessary to retrieve information from appropriate servents, to interpret it in a context and purpose-dependent way. Distributed User Modelling [10] addresses some of these issues, but they are currently out of the scope of this application.

4 Motivating Participation in COMUTELLA

By modeling the user, the servent can apply persuasion techniques through modifications of the interface or through rewarding the user with a better quality of service to attempt to influence the user's behaviour for the common good. Motivating users to participate is very similar to teaching them how to behave as good citizens. A basic principle of good teaching, leading back to Skinner, is to provide a plenty of positive feedback. It is important to reward users for good behaviour and not to give them the feeling that they are "punished" for bad behaviour (at least not in the beginning), since they may withdraw entirely from the system. Of course, negative feedback should be present too, in carefully selected doses depending on the user's level of participation.

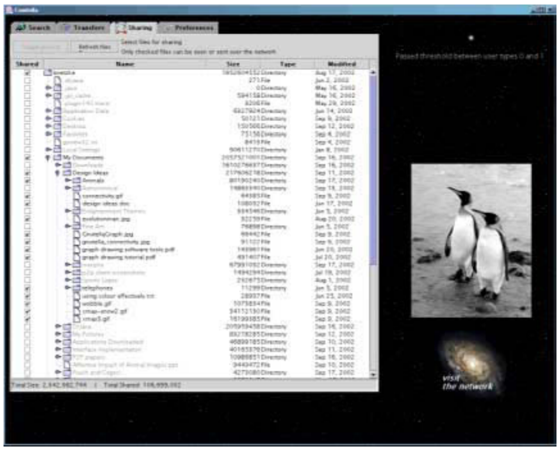
4.1 Motivating Altruistic Users

Altruistically motivated users are devoted to a particular cause (e.g. finding extra-terrestrial intelligence, cancer research or genome sequencing). They are likely to be active participants on higher levels (allow or create service) in an interest group dedicated to the cause, like SETI@home. Influencing people to be altruistic for a given cause is a very difficult task; it requires a very detailed and broad model of user interests and of her acquaintances in the real world (who might be involved in a interest group with a certain altruistic purpose). This seems still beyond the powers of the current user models and existing captology [4] (persuasion) techniques.

A simpler way that could hopefully influence the user is trying to invoke a feeling of guilt for not contributing to a community from which the user has taken a lot of resources. This could be attempted by using subtle cues like running messages in the window frame (should never be obtrusive), or by a face or animal figure that changes its expression with the change in the owing balance of the user to the group (see section 3.2.3). We have developed a simple iconic avatar that represents the servent (an eagle), which changes gradually to reflect the level of cooperativeness of the user towards a given community. This level is computed from the sum of balances of the user's relationships with the members of the community, and the user's personality characteristic from the user model. For each avatar there is a set of variants that differ in the level of friendliness of expression. Depending on the user's level of participation in the community, the avatar changes from a friendly sympathetic expression to an unfriendly and even vicious expression. This is accompanied with a running message (see Fig.3) in the window suggesting what the user can do to participate more actively in the community, depending on the current level of participation of the user. The idea is that, similar to Oscar Wilde's "The Picture of Dorian Grey" [12], the user will be cued to reflect on her social behaviour and how she can possibly change it for the better.

4.2 Motivating Reputation-Aware Users

Users motivated by social reputation are more likely to be active participants in groups where they already knows some participants (even if by alias) and are known themselves. This impact on the user can be achieved through an appropriately designed interface of the servent, that creates a global view of the group visualizing in an appropriate way the servents that contribute most. We are currently developing a background image of the servent for this purpose. It is inspired by the idea of a night sky where servents are represented as stars varying in size and brightness (see Fig.4). The size denotes the amount of resources or services shared by the servent and the brightness denotes the amount of relationships that the servent has with other servents. The user can access an annotated version of the image, and by positioning the mouse on a star the user can see the name / alias of the servent. The star representing the servent of the user is indicated, so the user can compare her significance in the community by the size and brightness. The image is refreshed periodically to reflect changes in the group.

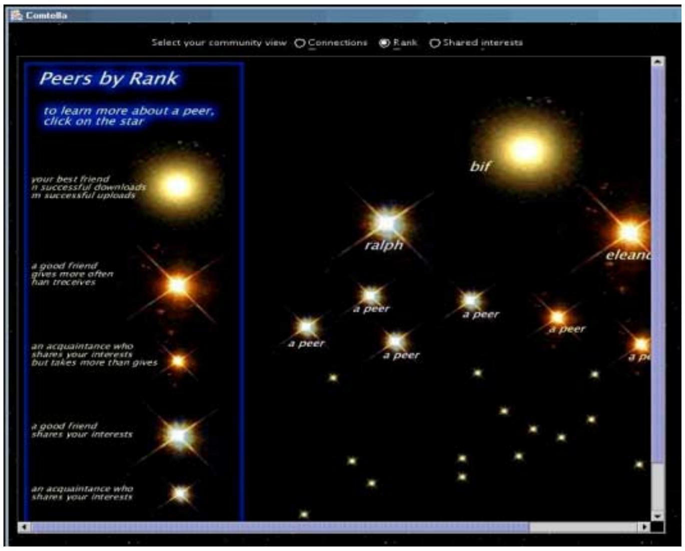


Select files for sharing

Only checked files can be seen or sent over the network

Passed threshold between user types 0 and 1

Fig 3. The motivational interface in COMUTELLA



Select your community view ○ Connections ● Rank ○ Shared interests

Comtella
Peers by Rank
to learn more about a peer, click on the star
your best friend n successful downloads m successful uploads
a good friend gives more often han treceives
an acquaintance who shares your interests but takes more than gives
a good friend shares your interests
an acquaintance who shares your interests
bif
ralph
eleand
a peer
a peer
a peer
a peer
a peer
a peer
a pe

Fig. 4. Visualizing the importance of servents (users) in the community

4.3 Creating a Feeling of Community

The feeling of belonging in a community motivates many people. They want to be known in the community, to find friends and people with similar tastes and values. While not always altruistic, these people tend to be altruistic in groups, which they perceive important for them (e.g. family, work-colleagues, neighbourhood). The forms of the altruistic behaviour and the level of contribution expected varies depending on the community.

More recently, approaches that try to exploit the social interactions between peers have been proposed. Local search strategies introduced in [1] use well-connected servents and have costs that scale sub-linearly with the size of the network. Ramanathan et al. [8] propose to dynamically determine the set of neighbors among those that have most related files (hits), thus indirectly defining a new neighborhood of each servent according to the current search. This reduces the number of query messages sent in the network by both reducing the number of servents to which a query is broadcasted and the TTL of each query). This approach provides mechanisms for servents in P2P systems to self-organize into groups.

In COMTELLA, friends of the user are treated differently by her servent. The servent ranks the requests coming from other servents depending on the balance and the importance of the relationship. In this way, in a download queue, priority or more bandwidth will be given to request from important servents to the user, or to servents, from who the user has downloaded often resources or whose services were often used. A servent X that "owes" to another servent Y can do it a "favour" by not decrementing the *time to live* (TTL, or the number of hops that the query can make) of a query sent by Y. In this way, the search horizon of a user who has contributed resources to users in the group increases.

Two users can be involved in relationships in several different contexts, indicating that they share interests in different areas. The relationships that the user maintains with other users in a given area of interest are sorted with respect to strength. From these, the top n ($2 <= n <= 10$) servents that are active at a given moment are used as immediate neighbours to send queries when the user starts a search with keywords from this area. In this way, when the user searches for resource in a given area, her servent asks first the "friends" of the user in this area. They are more likely to have resources in the area, since they have been involved in exchange in this area in the past. In this way, a user with many "friends" in different areas is in a better position to find resources.

4.4 Rewarding Participation

Several P2P systems (most prominently, Mojo-nation [11]) rely on a economic model to stimulate and reward participation. The basic assumption in the design of an economic model is that the effort and time spent bring new resources or services in the community have inherent costs. To take these costs into account, the resources/services should be made tradable. Thus the payment in a virtual currency (e-cash, duke-dollars, or mojo) may motivate a user to create new resources / services.

It is possible to reward the accumulated currency outside of the P2P system. For example, in the context of our system I-Help [5, 9], participation can be rewarded either in real dollars (for paid teaching assistants) or in marks (for students). The choice of reward depends on the type of the interest group. In Mojo-nation currency could be cashed in gift certificates from real-world vendors, though this wasn't very successful. It seems, that users are generally unwilling to participate in P2P systems with "micro-payments" [14], since the cognitive load to make the decision whether to request a resource/service outweighs the benefit of the service at the tip of your fingers. However, the users do not need to be aware that such micro-payments are happening. They can be arranged between the servents and the only thing visible to the user would be an improved or deteriorating Quality of Service (QoS) in terms of speed for locating and downloading resources.

Introducing micro-payments for resources creates a mechanism to balance the supply and demand of resources/ services and allows taking into account the different quality of resources or services provided [6]. Exchanging the accumulated currency in better quality of service can be a significant motivator and relatively easy to implement, because everything remains within the system.

In COMUTELLA we explore providing better Quality of Service (QoS) by allowing "richer" servents to "buy" themselves a wider the search horizon by negotiating the TTL of each query. Also, depending on the amount of "mojo" it is willing to pay for the service, the servent can jump to the top of the queue for a given service that is in a great demand at the moment. In this way users who have contributed to the community and earned a lot of "mojo" are able to gain a better quality of service, without even knowing that there is economy and currency involved.

If the accumulated currency is "cashed" in better QoS, is important to ensure a gradual improvement or decrement in the QoS depending on the level of user contribution (level of accumulated currency) and always to maintain clear cues in the interface as to what is the reason for the increase / decrease in the QoS and what the user should do in order to improve it. The QoS should not deteriorate completely even for uncooperative free riders, since there is always a hope that they may become cooperative, if they find the right interest group.

An economic model in a P2P system, however, brings an overhead: it requires additional reasoning capabilities on behalf of the servents/ servents, reduces the anonymity in the system [7], and requires centralized components to be introduced to be responsible for the currency transactions (as in Mojonation).

5 Conclusions

We propose a variety of methods to motivate users to participate actively in a P2P community. These methods rely on a servent which maintains a user model, which allows it to know an aspect of the personality of the user, the areas in which the user is interested and the friends or the most important relationships of the user in each area of interest. Such a servent will motivate the user in several ways: by interface cues targeted at provoking reflection in the user, by enforcing the user's feeling of being a part of a community through visualizing the standing of the users in the community, and by providing a better quality of service through a more informed search exploiting the social relationships that emerge between the user and her peers and implicitly creating user groups.

Unlike the approach proposed in [8], where the interest groups of users are highly dynamic and change rapidly to reflect the current search performed by the user, our approach relies on the assumption that users have long-term interests and are likely to search repeatedly in the same area at different times. Therefore, it makes sense to keep track of all "interest groups" of the user, to be able to use them again when a new search happens.

In addition, our approach relies on the assumption that the topology of the network in established active interest groups does not change too rapidly. We believe, that even though one of the biggest strengths of P2P systems is the ability to work in a highly dynamic environment, where servents (e.g. users) can come on-line and leave at any time, there is a pattern of behaviour that can be tracked (locally, by the individual servents) and adapted to, for the benefit of the users. The strength of the relationship between two users reflects not only a certain similarity in tastes and interest, but also a compatible pattern of being on-line. Users, who are related with strong relationships, who have been able to share valuable files / services in a mutually convenient time in the past, are likely to be able to do this again in the future.

The idea of flexibly changing the horizon for search has too been proposed in [8], depending on the how promising is the immediate neighbour to whom the request is sent. We extend this idea with the possibility of negotiating the search horizon between the servents and deploying currency (a measure of the user's cooperativeness, or balance of the relationship with the servent - e.g. if a the servent owes to our servent, it will let the query pass without decrementing the TTL). In contrast with [8], where goal is to reduce network traffic, our goal is to ensure better quality of service to users who contribute to the community.

Our future steps are evaluating the advantages and disadvantages of the proposed design with respect to user satisfaction and performance.

Acknowledgements

The author thanks Helen Bretzke, an NSERC / CRA-W summer student who investigated various persuasion strategies and designed the Comutella's interface. Thanks go also to Christopher Cox, an NSERC summer student for the implementation of the Comutella servent, and to Yamini Upadrashta, M.Sc. student for the implementation of the user modeling part of the servent.

References

[1] Adamic L. (1999) The Small World Web. Proc. 3rd European Conf. Research and Advanced Technology for Digital Libraries, ECDL http://citeseer.nj.nec.com/adamic99small.html

[2] Adar E., Huberman B. (2000) Free Riding on Gnutella. First Monday, vol. 5, no. 10. Also available on line at: http://www.firstmonday.dk/issues/issue5_10/adar/

[3] DAML Services: http://www.daml.org/services/

[4] Fogg B.J. (1998) Persuasive Computing: Perspectives and Research Directions, Proceedings CHI'88, http://hci.stanford.edu/captology/Key_Concepts/Papers/papers.html

[5] Greer J., McCalla G., Vassileva J., Deters R., Bull S., Kettel L. (2001) Lessons Learned in Deploying a Multi-Agent Learning Support System: The I-Help Experience, Proceedings of AI in Education AIED'2001, San Antonio, IOS Press: Amsterdam, 410-421. Available on line at: http://julita.usask.ca/homepage/public.htm

[6] Golle Ph., Leyton-Brown K., Mironov I. (2001) Incentives for Sharing in Peer-to-Peer Networks. Proceedings EC'01, October 12-17, 2001, Tampa, Florida, ACM press, 264-267

[7] Milojicic D., Kalogeraki V., Lukose R., Nagaraja K., Pruyne J., Richard B., Rollins S., Xu Z. (2002) Peer to Peer Computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto

[8] Ramanathan M. K., Kalogeraki, V. Pruyne J. (2001) Finding Good Peers in Peer-to-Peer Networks. Technical Report HPL-2001-271, HP Laboratories Palo Alto

[9] Vassileva J., J. Greer, G. McCalla, R. Deters, D. Zapata, C. Mudgal, S. Grant (1999) A Multi-Agent Approach to the Design of Peer-Help Environments, in Proceedings of AIED'99, Le Mans, France, July, 1999, 38-45. Available on line at: http://julita.usask.ca/homepage/public.htm

[10] Vassileva J. (2001) Distributed User Modelling for Universal Information Access, C. Stephanidis (ed.) "Universal Access in Human - Computer Interaction (UAHCI)", Proceedings of the 9th International Conference on Human-Computer Interaction, New Orleans, USA, vol.3, Lawrence Erlbaum: Mahwah, N.J., 122-126. Avaliable on line at:
http://julita.usask.ca/homepage/Agents.htm - MAS

[11] Wilcox-O'Hearn B. (2002) Experiences Deploying a Large-Scale Emergent Network, in Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)
http://www.cs.rice.edu/Conferences/IPTPS02/

[12] Wilde O. (1891) The Picture of Dorian Gray
http://www.bibliomania.com/0/0/57/103/frameset.html

[13] Yu, B., Singh. M. (2002) Emergence of Agent-Based Referral Networks. Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: part 3, July 2002, Bologna, July 15-19, 2002

[14] Shirky C. (2000) In Praise of Freeloaders, The O'Reilly Network. Available on line at:
http://www.oreillynet.com/pub/a/p2p/2000/12/01/shirky_freeloading.html

ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering

Carole Bernon[1], Marie-Pierre Gleizes[1], Sylvain Peyruqueou[2], and Gauthier Picard[1]

[1] Institut de Recherche en Informatique de Toulouse
118 route de Narbonne, 31062 Toulouse Cedex 4, France
{bernon,gleizes,picard}@irit.fr
[2] Artal Technologies
Parc Technologique du Canal, 31520 Ramonville Saint-Agne, France
peyruqueou@artal.fr

Abstract. Adaptive software is used in situations where either the environment is unpredictable or the system is open. This paper presents a methodology named ADELFE, which is led by the Rational Unified Process (RUP) but is devoted to software engineering of adaptive multi-agent systems. ADELFE guarantees that the software is developed according to the AMAS theory[1]. We focus this presentation on the additions of ADELFE regarding the three first core workflows of the RUP. Therefore, during the requirements phase, the environment of the studied system must be defined and characterized. Then, in the analysis phase, the engineer is guided to decide to use adaptive multi-agent technology and to identify the agents through the system and the environment models. Finally, the design workflow of ADELFE must provide the cooperative agentís model and helps the developer to define the local agentsí behavior. We illustrate the methodology by applying it to a case study: a timetable design.

1 Introduction

Today applications are more and more complex and a possible solution to deal with this complexity is the use of agent-based or multi-agent based software. Usually this kind of software considers an environment where all is predictable. However, applications are more and more dealing with environments that are unpredictable and submitted to changes like the Internet or the real world in which robots can evolve. They require more robustness, more autonomy, more complexity; they require adaptive software. In [13], according to the DARPA definition, it is said ìself-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possibleî. The multi-agent community

[1] The AMAS theory is developed and applied for the last 8 years at the Research Institute in Computer Science in Toulouse (IRIT). See http://www.irit.fr/SMAC

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 156-169, 2003.

generally considers the Multi-Agent Systems (MAS) as being adaptive because agents are autonomous, situated, pro-active, socialÖ The condition for the system to adapt is to be composed of autonomous agents. In our point of view, this kind of adaptation is weak adaptation and all existing multi-agent systems are adaptive in this sense.

We call ìstrong adaptationî of a multi-agent system, the ability this system possesses to take into account unpredictable events in order to react to evolutionary environments in order to realize its ìrightî task. This new generation of MAS represents the next challenge in programming. To theoretically study MAS with strong adaptation capability, we developed an Adaptive MAS (AMAS) theory [5][8].

The aim of this paper is then to present a methodology, named ADELFE, able to guide and help a designer when he wants to build an adaptive multi-agent system based on the AMAS theory, presented in section 2. An important point is that the agents considered in this theory are not adaptive, only the whole system is. In the third section of this article, a brief overview of ADELFE is given. In the section 4, ADELFE is applied to a timetabling case study to present in a more detailed way the different steps of the methodology. Before concluding this document, the main strengths and limits of ADELFE and its contributions, with regard to other methodologies, are given in the section 5.

2 The AMAS Theory

The first aim of this theory is to realize MAS having the ìclassicalî characteristics given in [7] to build a society of situated agents. But, from our point of view, such a MAS is also plunged into an environment and must reach a behavioral adequacy (by reproducing the behavior of a simulated society) or a functional one (by performing the right task, the task for which the system has been built). In our vision, the important notion is the collective; the AMAS theory must then lead to a coherent collective activity that realizes the right task.

We then proved the following theorem [8]: ìFor any functionally adequate system, there is at least a cooperative interior medium system which fulfills an equivalent function in the same environmentî. Therefore, we focused on the design of cooperative interior medium systems in which agents are in cooperative interactions.

The specificity of the theory resides in the fact that we do not code the global function of the system within the agents. The global function of this system emerges from the collective behavior of the different agents composing it. Each agent possesses the ability of self-organization i.e. the capacity to locally rearrange its interactions with others depending on the individual task it has to solve. Changing the interactions between agents can indeed lead to a change at the global level and this induces the modification of the global function. This capacity of self-organization at the lowest level enables to change the global function without coding this modification at the upper level of the system. Self-organization is founded on the capacity an agent possesses to be locally ìcooperativeî, this does not mean that it is always helping the other ones or that it is altruistic but only that it is able to recognize cooperation failures called ìNon Cooperative Situationsî (NCS, which could be related to exceptions in classical programs) and to treat them. The local treatment of NCS is a means to build a system that does the best it can when a difficulty is encountered. Such a difficulty is primarily due to the dynamical nature of the environment of the system, as well as the dynamics of the interactions between agents. More precisely an agent can detect three kinds of NCS:

- when a signal perceived from its environment is not understood and not read without ambiguity;
- when the information perceived does not induce the agent to an activity process;
- when concluding results lead to act in a useless way in the environment.

The agents, called cooperative agents, we are considering to build AMAS are composed of five parts contributing to their behavior: skills (what the agent is able to do), representations of the world (the knowledge it has about itself, about the others or about its environment), an interaction language (to communicate with others or with its environment), aptitudes (the capacities it possesses to reason on its knowledge) and a social attitude led by what we call cooperation.

Any designer needs a methodology to guide him when building a complex application. In the multi-agent domain, some methodologies exist [9][16] but few of them are able to deal with notions such as dynamics or evolving environment. This is the main reason why, from our point of view, a methodology suited for AMAS design is required. Furthermore, we want this methodology used by any designer and not only by those specialized in adaptive multi-agent systems because this would be a means to popularize this kind of programming.

3 A Short Overview of the ADELFE Methodology

This section highlights how ADELFE differs from object-oriented and other agent-oriented existing methodologies and where and how it handles the strong adaptation of the system to an evolutionary environment.

ADELFE[2] [1] is based on object-oriented methodologies, follows the Rational Unified Process (RUP) [10] and uses UML and AUML [12] notations. Some steps had been added in the classical workflows to be specific to adaptive MAS. It is not a general methodology such as GAIA [17] or TROPOS [6] but it has a niche which concerns applications that require adaptive multi-agent system design using the AMAS theory. ADELFE covers the entire process of software engineering like MESSAGE [4], PASSI [3] and TROPOS. In this paper, we only focus on the three first workflows, as shown in figure 1. At each workflow, the designer could backtrack to previous results to update or to complete them.

The **requirements workflow**, which is also taken into account in TROPOS, is a fundamental step in software engineering. In the AMAS theory, the adaptation process starts from the interactions between the system and its environment. Therefore, it is important to give a model of the environment during this workflow.

[2] ADELFE is a national project started in December 2000 and supported by the French Ministry of the Economy, Finance and Industry. The different partners of this project are IRIT (University of Toulouse III) and L3I (University of La Rochelle) from academia and are ARTAL and TNI from industry.

The **environment model** consists in three steps: determining the actors, defining the context and characterising the environment.

In the **analysis workflow**, we added two steps to the RUP: the identification of the agents and the adequacy of the AMAS theory. ADELFE focuses on the **identification of the agents** like in AAII [11], MESSAGE and PASSI. In the previous workflow, the designer has identified the entities of the system, now ADELFE must help him to identify what entities will be agents. In ADELFE, the notion of agent is restrictive; we are only interested in finding cooperative agents as described in the previous section. The designer has some guidelines: an agent is an entity previously defined, this entity may be faced with unexpected events and it may have evolutionary representations about itself, other agents or about its environment and/or evolutionary skills.

Because an adaptive MAS is not a technical solution for every application, ADELFE is the only methodology providing a tool to help a designer to decide if **the AMAS theory is adequate** to implement his application. For example, if the algorithm required to resolve the task is already known, if the task is not complex, if the system is closed or if no unexpected events can occur, this kind of programming is useless.

In the **design workflow**, the **agent model** and the **NCS model** are added to the RUP. The AAII methodology is dedicated to BDI agents, ADELFE is also dedicated to a specific architecture of agent: cooperative ones. Using the agent model, the designer must then describe the architecture of a cooperative agent by giving the components realising its behaviour. A MAS which is not in a cooperative interaction with its environment needs to adapt itself to it. But, according to the AMAS theory, the global function of the system is not coded, only the local behaviour of the agents composing the society is coded. The adaptation will then be managed by these agents through the NCS model. An agent which locally detects cooperative failures acts to change its interaction with others to remove this state. The NCS of an agent must be described by the designer, they depend on the application. To help him, ADELFE provides the designer with generic cooperative failures such as incomprehension, ambiguity, uselessness or conflict. ADELFE also provides tables with fields to fill up concerning the name of the NCS, its generic type, the state in which the agent must be to detect it, the conditions of its detection and what actions the agent must perform to treat it.

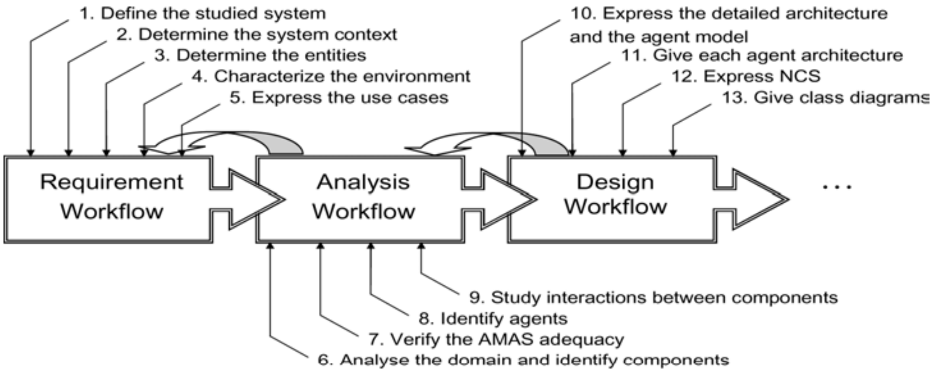

1. Define the studied system
2. Determine the system context
3. Determine the entities
4. Characterize the environment
5. Express the use cases
10. Express the detailed architecture and the agent model
11. Give each agent architecture
12. Express NCS
13. Give class diagrams
Requirement Workflow
Analysis Workflow
Design Workflow
...
9. Study interactions between components
8. Identify agents
7. Verify the AMAS adequacy
6. Analyse the domain and identify components

Fig. 1. Overview of the three first core workflows of ADELFE

4 Step-by-Step Details of the Methodology

In this section, we present the different steps of the ADELFE methodology. A case study about timetabling problem[3] illustrates the methodology at each step. The main actors of this case study are teachers and students who need to book several rooms to achieve theirs tasks. This case study has been chosen because it is based on a non-predictable environment and it is an open problem. Actually, some rooms could become unavailable because of some kind of problem or a teacherís availability could change and giving a solution to this problem means adapting in order to react to these dynamic changes of the environment. Furthermore, this problem can be extended to a diary management, for example, in which new people, and so new agents, must be added in the running system; this problem can be considered are being open.

4.1 Requirements Workflow

The aims of this stage are to define the system to be, to transform this view in a use-case model, and to organize and to manage the requirements (functional or not) and their priorities. In fact, at this stage, the designer has to define the function of the studied system and to model its environment.

Definition of the studied system. This definition is the result of the analysis of the requirements set artifact. The output of this step is a keyword set defining the system. From the user requirements, for the timetabling problem, the following keywords and concepts are highlighted: planning, rooms, teachers, students, constraints, organization and constraint managing. As an outlook, the problem can be viewed as the organization of teachers and students in rooms (and all the participants are subject to constraints). This problem belongs to the constraint satisfaction problems class.

Definition and Environment Modeling. A detailed definition of the environment of the system is necessary to develop adaptive systems, which are able to respond to any change. This step firstly focuses on what may be in interaction with the studied system in terms of passive or active entities, or constraints. In our example, teachers, students, the planning manager and the room manager are active entities because they are able to change by themselves their own constraints or they can interact with the system. Rooms are passive because they represent resources and they cannot modify their characteristics by themselves. The PPN (or National Pedagogic Plan) is the database that contains all the information concerning the courses (maximum number of sessions per week, hour quotas for each formation, etc): it is a passive entity.

In a second time, this step must define the context of the system. It requires a characterization of data flows and interactions between passive or active entities and the system. This step produces collaboration diagrams and sequence diagrams (entity/system or entity/entity). In our example, there are two kinds of data flows between the system and passive entities: when the system consults the PPN and when the system consults room constraints. When an active entity wants to interact with the

[3] We elaborated this example as a case study to compare and discuss different methodologies and multi-agent platforms for the ASA Group of the Artificial Intelligence French Association.

system, it may only have to change constraints (owner constraints or room constraints). In the other sense, the system interacts with the active entity by displaying the planning.

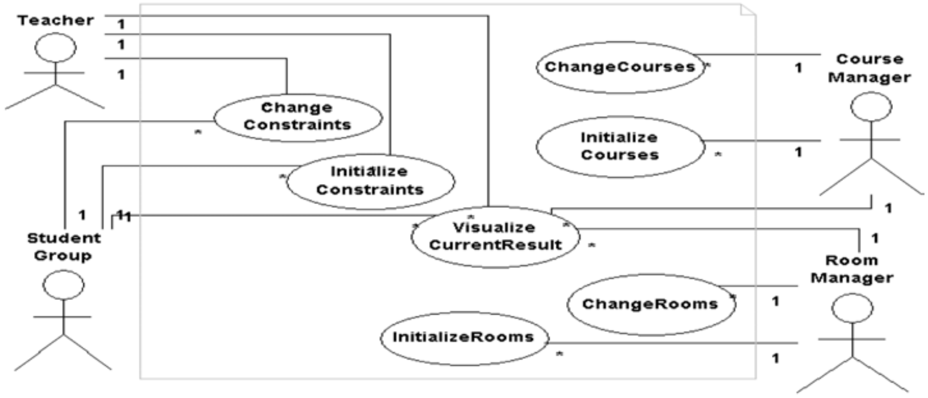


Fig. 2. The use cases and their related actors for the timetabling problem

The third stress of this step is to characterize the environment according to the classification given by Russell in [14]. This characterization may enable the designer to detect some special use cases that aim to respond to the environment behavior. In the case study, we can characterize the environment of the system as following:

- Dynamic: the evolution of the active entities does not depend on the system, it is unpredictable from the point of view of the system;
- Accessible: the system can obtain information on the state of the environment;
- Non-deterministic: the system is not able to know what could be the effects of its actions on the active entities;
- Continuous: the number of interactions between the system and the entities is infinite.

Determination of the Use Cases. The main objective of this step, which ends the requirements workflow, is to clarify the different functionalities the system has to respond to. Only the active entities are implied in these use cases, which are the results of a functional requirements set. The use cases for the timetabling problem are shown in figure 2.

4.2 Analysis Workflow

From a multi-agent point of view, the identification of the agents must take place in this workflow. The analysis workflow has to develop an understanding of the system, its structure in terms of components and to know if the AMAS theory is required.

Domain Analysis and Architecture Study. Domain analysis is a static view and an abstraction of the real world and the linked entities. Considering separately each use-case by defining scenarios, the designer has to divide the system into entities. The result of this step is a set of entities in preliminary class diagrams. `Teacher`, `CourseManager`, `StudentGroup`, `Room`, `RoomManager` and `PPN` classes appear naturally as real world entities. In a second time, we tried to determine what entities could be useful for our system. We propose a board to visualize the organization (the `Grid` and `Cell` classes) and the `ConstraintManager` class to control constraints for each entity that owns a `Constraint` class instance. Cells represent intersections of different dimensions (days, rooms, etc).

Adequacy of the AMAS Theory. This step aims to help the designer to decide if the AMAS theory is adequate to solve his problem because, for certain applications, this kind of programming can be useless. A software has been developed with several criteria to study the adequacy at two levels:

- At the global level to answer the question ìis a system implementation using AMAS needed?î
- At the local level to try to answer the question ìdo some components need to be implemented as AMAS?î i.e. is some decomposition or recursion useful during design?

For the case study, the decision tool clearly suggests to use the AMAS to design the global level. Moreover, the tool indicates that some entities could be decomposed as AMAS. So, once the agents are identified, the designer has to reuse the method on them, as developed below.

Agent Identification. In this step, we are only interested in agents that enable a designer to build our sort of AMAS. The designer has to determine which entities fit with this agent type. This identification is done considering the notion of cooperative agent and the agentís characteristics such as autonomy, locality, requirement of interactions, individual goal to achieve, capacity of negotiation. Firstly, we have to know where a lot of evolution or dynamic is required. Then, for each entity identified during the domain analysis, we must examine if it has to face up to unpredictable events and has to treat Non Cooperative Situations. Teachers and students are autonomous, have local views, are plunged in the rooms and have to negotiate to find partners and to resolve resource problems. This kind of situations may create cooperation failures (NCS). At this stage, we identify teachers and students groups as being cooperative agents. All other entities are considered as objects.

Adequacy of the AMAS Theory at the Local Level. If the first step of adequacy to the AMAS theory indicates a possible decomposition, each agent has to be analyzed as a system. The goals of an agent, `Teacher` or `StudentGroup`, are to find different places and partners to follow or to give each course. These goals raise the problem of ubiquity. Agents cannot be at different place at different moments. Therefore, we propose to create one agent per course for each teacher or student group. Two agent levels are distinguished:

- `RepresentativeAgent` (RA): at the highest level, it represents a teacher or a student group within the system;
- `BookingAgent` (BA): at the lowest level, it is responsible for finding partners and booking rooms for a `RepresentativeAgent`. There are as many BAs as the number of courses a teacher has to give or a student group has to follow.

The identified agents have to be added to the preliminary class diagram as shown in fig. 3.

Study of Interactions between the Different Entities. The result of this step is a set of sequence diagrams and activity diagrams, which explains the possible interactions between the different entities within the system and at each level. As the RUP is use-case guided, for each use case, which has been defined between the system and the environment, a sequence diagram has to be defined to show the internal view and the interactions within the system. The interaction between agents can be written with AUML.

4.3 Design Workflow

In this section, we detail the design workflow in four steps. Because the complete design cannot be described in this paper, we only detail the steps which are not existing in other methodologies such as the agent model and the modeling of Non Cooperative Situations.

Detailed Architecture and Agent Model. The first step of the design requires to identify the software components and to describe them. The result provides the architecture of the system in terms of needed blocks, classes, agents and interactions. The agent model, which represents the relationships between agents, is included in this architecture. The previously defined architecture can be refined by determining if some design patterns and/or re-usable components can be used. For example, in object-oriented methodologies designers try to re-use models such as customer-server modelÖ In ADELFE, we propose a specific design pattern named cooperative agent architecture. In our case study, four packages appear:

- Agent package, to manage BAs and RAs;
- Grid package, to manage the different dimensions of a grid and its cells;
- Constraint package, which has to be accessible to rooms and agents;
- Interface package, to enable a user to interact with the system.

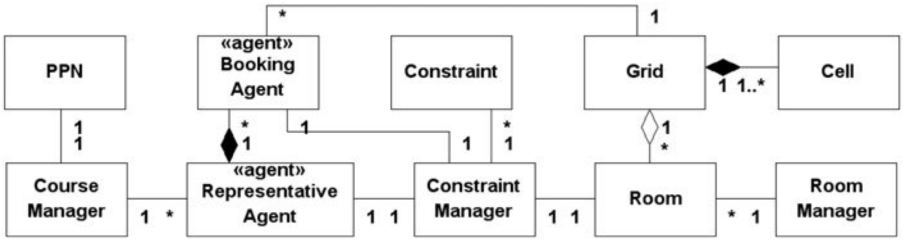


Fig. 3. The revised preliminary class diagram for the timetabling problem gives us an idea of the MAS model

Table 1. `BookingAgent` specification: skills, aptitudes and interaction languages are methods; representations are attributes

BookingAgent	
Representations	
• Constraints • bookState • partnershipState • brotherConstraints	• partners • recentlyMetAgents • RAFather • currentCell
Skills	
• moveInTheGrid • manageConstraints • manageBooking	• managePartnership • manageMessages
Aptitudes	
• bookARoom • cancelBooking • negotiateBooking • establishPartnership	• cancelPartnership • negotiatePartnership • SendMessage • InterpretMessage
Interaction Language	
• messageInteraction	• contactInteraction

Cooperative Agent Architecture. This step helps the designer to fill in a generic architecture given for an agent used in the AMAS theory. This architecture can be considered as a design pattern supplied by ADELFE in which an agent is composed of a skill model, representations models, an interaction language, aptitudes and a social attitude named cooperation. Following an object-oriented methodology, skills, representations or aptitudes possessed by an agent could be represented by methods. However, one of the main characteristics of the agents used in AMAS is the fact that sometimes they must be able to learn new skills, new representations or new aptitudes to adapt themselves to an evolutionary environment. Methods cannot change as the time goes by, so, if these characteristics are dynamic, they must adopt new representations: they must be AMAS themselves. In such a case, ADELFE recommends the designer to apply again the methodology to realize an adaptive multi-agent system to implement the skills, the representations or the aptitudes of an agent. The interaction language can be implemented by a set of classes or by a design pattern, which is closed to specific communication tools between agents like ACL implementation. Finally, the social attitude of an agent is guided by cooperation, it must be able to solve Non Cooperative Situations and a model is purposely defined in the following step. An example for a `BookingAgent` is given in the table 1. A BA is able to move in the grid, to meet other BAs and to negotiate partnerships. It can also send and receive messages and interpret them. To achieve its task, it must know the constraints of all the BAs working for its `RepresentativeAgent`. Constraints can be relaxed during a negotiation.

Table 2. The Booking Conflict NCS table of a `BookingAgent`

Booking Conflict
State
Any
Description
The BA is in a cell that is interesting to book but this cell is already booked
Conditions
The BA is in a cell **AND** this latter is already booked **AND** yet the cell would be suitable if not booked
Actions
IF the cost of the new booking is less than the older one **THEN** the BA books the cell **ELSE** the BA moves elsewhere

Non Cooperative Situation Model. This step represents the main contribution of ADELFE to this workflow. During it, the designer must fill up a table describing each NCS encountered by each previously identified agent. This table contains the name of the NCS, the state in which the agent is when detecting it, the textual description of the NCS and the conditions and actions linked to it. The conditions describe the different elements that enable the agent to locally detect this NCS. The actions describe what the agent has to do to remove it. For instance, the NCS for a `BookingAgent` are:

- *Partnership incompetence*: the BA meets another BA that may be an uninteresting partner;
- *Booking incompetence*: the BA is in a cell that is uninteresting to book;
- *Message unproductiveness*: the BA receives a message that is not correctly addressed;
- *Partnership conflict*: the BA meets another BA that is interesting, but this other BA has already a partner;
- *Booking conflict*: the BA is in a cell that is interesting to book but this cell is already booked (shown in table 2);
- *Booking uselessness*: the BA meets its partner: they must separate to explore more efficiently the grid.

Even if all the behaviors of the cooperative agents are given (agent model + NCS model), the MAS as a whole can adapt itself because the interactions are not a priori coded. Changing the interactions between agents (self-organization) changes the global function of the system and, then, allows the ìstrong adaptationî of the MAS.

Class Diagrams. This step provides the different class diagrams for taking the GUI and database designs into account. These class diagrams are also refinements of the previous class diagrams. ADELFE does not need to provide additional functionalities to build them because the designer follows the same method as the one used in an object-oriented methodology. The figure 4 shows the final class diagram for the system.

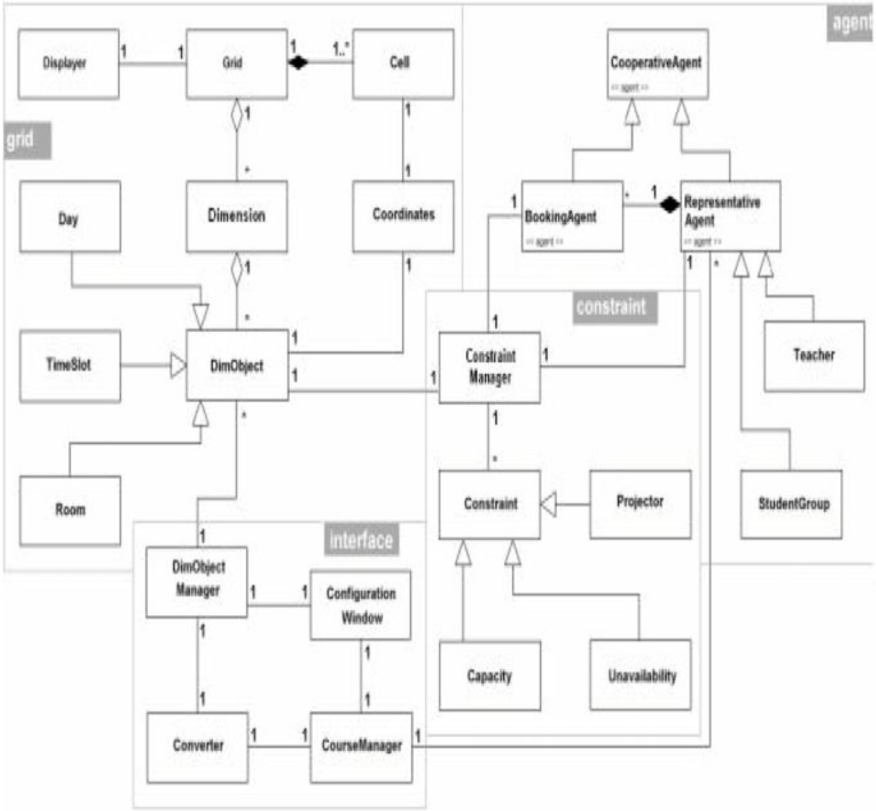


Fig. 4. The final class diagram for the timetabling problem and the static inter package links between agents and objects

4.4 Implementation and Tests Workflows

This workflow is similar to what is done in the RUP. However, to facilitate the work of the developer, we plan to provide a rapid prototyping tool that will be integrated in the OpenTool© software provided by our partner TNI (http://www.tni.fr). OpenTool is a graphical tool like Rational Rose, which supports the UML notation.

Concerning the timetabling problem, we have implemented the proposed architecture. Agents find adequate organizations and provide relevant results. At the moment, we only have tested the system on simple examples that appear in the requirements, but we hope to enlarge its application.

5 Contributions, Strengths and Limitations of ADELFE

TROPOS [6] expresses the dynamic and openness of the application in the requirements phases with the model of the environment and with particular soft goals. However, it does not give guidelines to design the right agentsí behavior allowing the adaptability of the system. GAIA indicates that the domain covered by the methodology is static and that the methodology is dedicated to closed domain where agentsí skills and beliefs are static at run-time [16]. However some improvements to GAIA are suggested in [18] to support applications in dynamic domains. Many other methodologies, like AAII [11], MaSE [15] or MESSAGE [4], do not focus on the dynamic aspect of the software environment and on the adaptation abilities of the software.

The specificity of ADELFE is to provide a methodology to design adaptive multi-agent systems coupled with a theory on those systems. The provided methodology covers the whole ìclassicalî life cycle of software and reuses UML notations. These notions are already familiar to most of designers making this tool easily usable by developers not specialized in designing AMAS.

In adaptive multi-agent systems, the environment (in which the system is operating) is a key notion; but in a general way, the environment modeling is not a central point in existing methodologies. In DESIRE [2], the environment is taken into account at the agent level in the ìworld interaction management moduleî: an agent maintains and interacts with its environment in the same way as with other agents. In MaSE, GAIA and AUML there is no particular model of the environment. In TROPOS, the environment model is described in terms of actors, their goals and interdependencies. In MESSAGE, the domain model captures some entities of the system environment and the interactions with the environment are described for each role in terms of sensory inputs and acquaintances, resources ownership and accesses, and finally tasks and actions. In AAII, the relation between the agent and the environment is taken into account in the interaction model.

For an industrial, it is very important to know very early if the system to develop justifies some investment in a new methodology or technique. Therefore, ADELFE guides the developer to make him decide if and where the adaptive multi-agent system technology is required in the system that he is developing. This explains the importance of the adequacy checking in the analysis stage.

ADELFE helps the designer to find what components of his system demand to be treated like agents belonging to the AMAS theory (cooperative agents). The specific agent architecture recommended by ADELFE can be viewed as one more design-pattern provided to the developer. Furthermore, ADELFE guides him to build agents; it provides a functionality to endow an agent behavior with the NCS model. This is because it is well known that the autonomous behavior of an agent which results from its perceptions, its knowledge and its beliefs is very difficult to define in complex systems as well as in dynamic systems. Actually, it is very difficult to enumerate all possible actions for each state of the environment. ADELFE is also a recursive or iterative methodology, when the developer has to implement an agent he must reuse the entire methodology to develop it.

ADELFE does not allow the design of every agent-based application. This drawback can be taken away by coupling another methodology (such as MESSAGE or PASSI) with ADELFE. If the adequacy phase tells that adaptive systems are not necessary, another methodology could be proposed. There is no automated tool for consistency checking of the workflows results but it is a future improvement.

6 Conclusion and Future

Few of the existing agent-oriented methodologies deal with concepts like dynamism or evolving environment. The aim of this paper was then to present the ADELFE methodology which is a multi-agent-oriented methodology suited to adaptive multi-agent systems based on the AMAS theory. ADELFE provides a new methodology to design a society of agents showing a coherent activity. It allows the society of agents to self-adapt to its environment. Till now, ADELFE has been used in two case studies: an Intranet system design [1] and a timetabling problem. In the former one, the requirements, the analysis and the design have been realized. In the later, the system is now implemented and operational. We plan to experiment more ADELFE in a European project (SYNAMEC) to develop a mechanical self-design system.

ADELFE is aimed to be a development toolkit of software with emerging functionalities and not only a ìmereî methodology. Therefore, we have some perspectives for it, ADELFE will be able to:

- Provide some tools and libraries to ease the design and development of systems. For example, we think that it is better for a designer to have the ability of rapid prototyping to judge the validity of his architecture;
- Assist the designer if another methodology is more adequate to the system he wants to build.

References

[1] C. Bernon, M-P. Gleizes, G. Picard & P. Glize ñ The ADELFE Methodology for an Intranet System ñ *Agent-Oriented Information Systems (AOISí02) at CAiSEí02*, Toronto, May 2002.

[2] F. M. T. Brazier, P.A.T. Van Eck & J. Treur - Modelling a society of simple agents: From conceptual specification to experimentation - *Journal of Applied Intelligence*, 14:161ñ178, 2001.

[3] P. Burrafato & M. Cossentino ñ Designing a Multi-Agent Solution for a Bookstore with the PASSI Methodology ñ *AOISí02 at CAiSEí02*, Toronto, May 2002.

[4] G. Caire, F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, G. Pavon, P. Kearney, J. Stark & P. Massonet - Agent Oriented Analysis using MESSAGE/UML - *AOSE 2001*.

[5] V. Camps, M-P. Gleizes, S. Trouilhet - Properties Analysis of a Learning Algorithm for Adaptive Systems ñ In *International Journal of Computing Anticipatory Systems*, Editions Chaos, LiËge, Belgium, 1998. Available at http://www.irit.fr/SMAC.

[6] J. Castro, M. Kolp & J. Mylopoulos ñ A Requirements-driven Development Methodology ñ In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSEí01)*, Stafford, UK ñ June, 2001.

[7] J. Ferber - Les SytËmes Multi-Agents. Vers une Intelligence Collective - InterEditions 1995

[8] M-P. Gleizes, V. Camps, P. Glize - A Theory of Emergent Computation Based on Cooperative Self-Organization for Adaptive Artificial Systems - *4th European Congress of Systems Science*, Valencia, 1999. Available at http://www.irit.fr/SMAC.

[9] C.M. Iglesias, M. Garijo & J. C. Gonzalez - A Survey of Agent-Oriented Methodologies - In *Intelligent Agents V, ATAL'98*, LNAI 1555, Springer Verlag 1999.

[10] Jacobson, G. Booch & J. Rumbaugh ñ *The Unified Software Development Process* ñ Addison-Wesley, 1999.

[11] D. Kinny, M. Georgeff, & A. Rao - A Methodology and Modeling Technique for Systems of BDI Agents - In W. Van de Velde and J.W. Perram, editors, *Agents Breaking Away: Proceedings of the 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World* (LNAI 1038), pp 56-71. Springer Verlag, 1996.

[12] J. Odell, H.V. Parunak, & B. Bauer - Extending UML for Agents - In Proceedings of the Agent Oriented Information Systems (AOIS) Workshop at the 17th National Conference on Artificial Intelligence (AAAI), 2000.

[13] P. Robertson, R. Laddaga & H. Shrobe - Introduction: the First International Workshop on Self-Adaptive Software - In *Proceedings of the 1st IWSAS* edited by P. Robertson, R. Laddaga & H. Shrobe in LNCS 1936, pp 1-10, 2000.

[14] S. Russel & P. Norvig - Artificial Intelligence: a Modern Approach - *Prentice-Hall*.

[15] C. H. Sparkman, S. A. Deloach, A. L. Self ñ Automated Derivation of Complex Agent Architectures from Analysis Specifications ñ *AOSE-2001*, Montreal, Canada, May 29th 2001.

[16] M. Wooldridge & P. Ciancarini - Agent-Oriented Software Engineering: the State of the Art - In P. Ciancarini & M. Wooldridge, ed., *Agent-Oriented Software Engineering*, Springer Verlag LNAI 1957, January 2001.

[17] M. Wooldridge, N. R. Jennings & D. Kinny - A Methodology for Agent-Oriented Analysis and Design - In *Proceedings of the 3rd International Conference on Autonomous Agents (Agents 99)*, pp 69-76, Seattle, WA, May 1999.

[18] F. Zambonelli, N. R. Jennings, and M. Wooldridge - Organisational abstractions for the analysis and design of multi-agent systems - In P. Ciancarini and M. Wooldridge, eds., *AOSEí00*, LNCS, Springer-Verlag, 2000.

Evaluating Multi-agent System Architectures: A Case Study Concerning Dynamic Resource Allocation

Paul Davidsson and Stefan Johansson

Department of Software Engineering and Computer Science,
Blekinge Institute of Technology, Soft Center, 372 25 Ronneby, Sweden
{pdv,sja}@bth.se

Abstract. Much effort has been spent on suggesting and implementing new architectures of Multi-Agent Systems. However, we believe the time has come to compare and evaluate these architectures in a more systematic way. Rather than just studying a particular application, we suggest that more general problem domains corresponding to sets of applications should be studied. Similarly, we argue that it is more useful to study the properties of classes of multi-agent system architectures than particular architectures. Also, it is important to evaluate the architectures in several dimensions, both different performance-related attributes, which are domain dependent and more general quality attributes, such as, robustness, modifiability, and scalability. As a case study we investigate the general problem of "dynamic resource allocation" and present four classes of multi-agent system architectures that solve this problem. These classes are discriminated by their degree of distribution of control and degree of synchronization. Finally, we instantiate each of these architecture classes and evaluate, through simulation experiments, how they solve a concrete dynamic resource allocation problem, namely load balancing and overload control of Intelligent Networks.

1 Introduction

Much effort has been spent on suggesting and implementing new architectures of Multi-Agent Systems (MAS). Unfortunately, this work has been carried out in a quite unstructured way where a (group of) researcher(s) invents a new architecture and applies it to a particular domain and conclude that it seems to be appropriate for this domain. Often this new architecture is not even compared to any existing architecture. We believe that this area now has reached the level of maturity when it is appropriate to compare and evaluate these architectures in a more systematic way.

1.1 Applications

Of course, there is no single MAS architecture that is the most suitable for all applications. On the other hand, to find out whether one architecture performs

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 170–183, 2003.

better than another for a particular application is usually of limited scientific interest. (Although this information may be very useful to solve that particular problem.) Instead, we suggest the study of more general problem domains corresponding to sets of applications with common characteristics. In this paper we will exemplify this approach by investigating the general problem of *dynamic resource allocation*. Such studies tend to be quite abstract and are for that reason often of a theoretical and qualitative nature. They therefore should be supplemented with and validated by quantitative empirical studies in one or more concrete applications corresponding to instances of the general domain, in this paper exemplified by load balancing and overload control in *Intelligent Networks*, a type of telecommunication system.

1.2 Architectures

Just as it is useful to study classes of applications rather than particular applications, we argue that it is useful to study classes of MAS architectures in addition to particular architectures. To make such studies possible, we need to describe MAS architectures in a way that abstracts the particularities of the individual architectures but still captures their relevant characteristics. To develop a general way of characterizing MAS architectures is in itself a major research task. In fact, it may be necessary to use several views to capture all relevant aspects of an architecture cf. Kruchten [9].

In this work we will categorize MAS architectures according to two properties: the type of control used (from fully centralized to fully distributed), the type of coordination (synchronous vs. asynchronous). As with classes of applications, it is mostly theoretical studies that can be performed on classes of MAS architectures. Therefore, they often need to be supplemented with empirical studies using instantiations of these architectures. Below we will present four concrete architectures corresponding to different combinations of the two architectural properties.

1.3 Quality Attributes

It is possible to evaluate MAS architectures with respect to several different quality attributes, both different performance-related attributes and more general quality attributes, such as, robustness, modifiability, and scalability. Some of these attributes are domain independent and some are specific for each set of applications, e.g., performance-related attributes. As mentioned earlier, we do not think it is possible to find a MAS architecture that is optimal with respect to all relevant attributes. Rather, there is an inherent trade-off between these attributes and different architectures balance this trade-off in various ways. The various applications, on the other hand, require different balances of this trade-off. Thus, in order to choose the right architecture for a particular application, knowledge about relevant attributes and how different MAS architectures support them is essential.

1.4 Evaluation Framework

To evaluate a set of architectures in a systematic way, we suggest an approach that can be described in terms of the following three-dimensional space:

- the set of possible applications,
- the set of possible MAS architectures, and
- the set of attributes used to evaluate the architectures.

The suggested approach is to investigate substantial parts of this space rather than just single points. We believe that this approach, besides of enabling a more systematic investigation of the space, will lead to a deeper understanding of MASs and their applications, which, in turn, will contribute to reach the long-term goal of obtaining general design principles of MAS.

In this paper we will apply this approach to the general problem of dynamic resource allocation and present four abstract MAS architectures with different characteristics that solves the problem. These will then be compared with respect to a number attributes, e.g., reactivity, ability to balance the loads, fairness, utilization of resources, responsiveness, amount of communication overhead, robustness, modifiability, and scalability. Finally, we evaluate concrete instantiations of these abstract architectures in a concrete dynamic resource allocation problem, namely load balancing and overload control in Intelligent Networks.

2 Abstract Domain: Dynamic Resource Allocation

Multi-agent technology has proved to be successful for *dynamic resource allocation*, e.g. power load management [11] and cellular phone bandwidth allocation [3]. Basically, this problem concerns the allocating of resources between a number of *customers*, given a number of *providers*. The dynamics of the problem lies in that the needs of the customers, as well as the amount of resources made available by the providers, vary over time. The needs and available resources not only vary on an individual level, but also the total needs and available resources within the system. We will here assume that the resources cannot be buffered, i.e., they have to be consumed immediately, and that the cost of communication (and transportation of resources) between any customer-provider pair is equal.

2.1 Abstract Multi-agent Architectures

There are many ways of dividing the set of possible MAS architectures into different subsets based on their characteristics, e.g.:

- the topography of the system,
- the degree of mobility and dynamics of the communications,
- the degree of distribution of control, and
- the degree of synchronization of interaction.

We have chosen to focus the two last properties. By degree of *distribution* we mean to what degree the *control* of the system is distributed. The degree of *synchronization* is a measure of how the execution of the agents interrelate with each other. We may have agents that are highly sophisticated, but who only interacts at special slots in time, and thus have a high degree of synchronization. There are also systems in which the agents may interact continuously, independently of when other agents interact, which we will refer to as *asynchronous*.

To sum up, we will compare the following four abstract classes of MAS architectures for dynamic resource allocation: centralized synchronous architectures, centralized asynchronous architectures, distributed synchronous architectures, and distributed asynchronous architectures.

2.2 Abstract Attributes

We have identified the following important performance-related attributes to dynamic resource allocation:

- Reactivity: *How fast are resources re-allocated when there are changes in demand?*
- Load balancing: *How evenly is the load balanced between the resource providers?*
- Fairness: *Are the customers treated equally?*
- Utilization of resources: *Are the available resources utilized as much as is possible?*
- Responsiveness: *How long does it take for the customers to get response to an individual request?*
- Communication overhead: *How much extra communication is needed for the resource allocation?*

In addition, there are a number of more general software architecture quality attributes [6] that should be addressed, e.g.:

- Robustness: *How vulnerable is the system to node or link failures?*
- Modifiability: *How easy is it to change the system after it is implemented (and often deployed)?*
- Scalability: *How good is the system at handling large numbers of users (providers and customers)?*

2.3 Theoretical/Qualitative Evaluation

We will now make a brief theoretical, or qualitative, analysis of how the degree distribution and synchronization of the multi-agent system architecture influence the attributes identified in the last section.

- Reactivity should be promoted by asynchronous architectures since there is no need to await any synchronization event before i) an agent can notify other agents about changes in demand and ii) other agents can take the appropriate actions to adapt to these changes.

- Load balancing should be favored, or at least not disfavored, by centralized control since it is possible to take advantage of the global view of the state of the system, e.g., the current load at the providers and the current demand of the customers.
- Similarly should fairness be easier to achieve for architectures with centralized control since they have information about the global state of the system.
- It is not clear from a strictly theoretical analysis if there is any correlation between the ability to utilize the resources and the architectural properties. Empirical studies are probably necessary.
- Also, it is not clear from a strictly theoretical analysis if there is any correlation between responsiveness and the architecture properties.
- Communication overhead can be measured either by the number of messages sent, or by the bandwidth required for the allocation. Synchronous architectures tend to concentrate the message sending to short time intervals, and thus requiring a large bandwidth, whereas asynchronous architectures tend to be better at utilizing a given bandwidth over the time. Also, communication in distributed architectures has a tendency to be more local than in centralized architecture, using smaller parts of the network.
- Regarding robustness we conclude that the more centralized the control is, the more vulnerable the system gets. Basically, the reason for this is that the system cannot function, i.e., perform reallocation, if the agents that are responsible for the control fail. In more distributed systems, the reallocation may function partially even though some agents have failed.
- The modifiability, to add or remove a provider or customer, seems to be better in centralized architectures. For instance, changes may only be necessary in one part of the system.
- Scalability seems to be better supported by distributed architectures than centralized architectures. Firstly, the computational load for the resource allocation is divided between a number of computers, and secondly, the risk for communication bottlenecks is smaller.

3 Concrete Domain: Load Balancing in Intelligent Networks

One important area in which the dynamic resource allocation problem is present is telecommunications. The Intelligent Network (IN) concept was developed in order to enable operators of telecommunication networks to create and maintain new types of services [10]. Two important entities of an IN are the Service Switching Points (SSPs) and the Service Control Points (SCPs). The SSPs continuously receive requests of services which they cannot serve without the help of the SCPs where all service software resides. Thus, the SCPs are providers and the SSPs are customers. The SSPs and SCPs communicate via a signaling network which we here will represent as a *cloud* rather than a specific topology of signaling links and nodes. (See Figure 1.) It is assumed that a small part of the available bandwidth of this network is reserved for the resource allocation, i.e.,

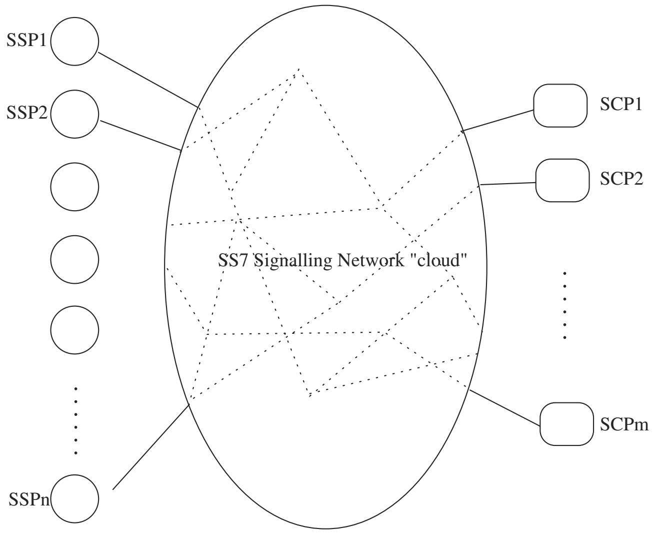


Fig. 1. A simplified view of an Intelligent Network (IN) with m SCPs and n SSPs (typically $n > m$)

the communication overhead caused by agent communication (and transportation). It is assumed that all SCPs support the same set of service classes and that all service requests can be directed by a SSP to any SCP.

3.1 Concrete Multi-agent System Architectures

We have chosen one architecture of each of the abstract classes mentioned earlier (see table 1). Common for these architectures are the use of three different types of agents: *quantifiers, allocators,* and *distributors*. A quantifier acts on behalf of a provider of the resources, an allocator acts on behalf of a customer, and a distributor decides the allocation of some (or even all) available resources. Although these three types of agents have similar roles in all the four multi-agent system architectures, the actual implementation may be rather different (in particular this hold for the distributors). The reason, of course, is that different system architectures may put different demands on the agents.

Table 1. The four different multi-agent system architectures classified in terms of distributedness and synchronicity

	centralized	distributed
synchronous	Centralized auctions (CA)	Hierarchical auctions (HA)
asynchronous	Centralized leaky bucket (CLB)	Mobile brokers (MB)

The Centralized Auction Architecture The *Centralized Auction* (CA) architecture is an example of a synchronous, centralized architecture. Arvidsson et al. [1] suggested an approach where the resource allocation is carried out by means of tokens (cf. market-based control [5]). Each token represents a service request and is consumed when the request is accepted by a provider. The three types of agents have the following functionality:

- The quantifiers try to *sell* the amount of tokens that corresponds to the load that the provider is able *serve* between two auctions.
- The allocators try to *buy* the amount of token corresponding to the resources it predicts their customer will receive during the time to the next auction.
- The distributor receives bids from the quantifiers corresponding the available capacity at their provider (and the *prices*), and bids from the allocators containing the expected need for resources. The distributor then carries out the auction so that the common good is maximized and sends messages about the result to the involved agents.

An allocator maintains a pool of tokens for each provider and type of resource. Each time the allocator feeds a provider with a request for a particular type of resource, one token is removed from the associated pool. If all pools associated with a particular resource type are empty, the customer cannot accept more requests. The pools are refilled at the auctions that take place at fixed time intervals. In order to avoid spending all tokens immediately during high loads (which would lead to excessive delays caused by long queues at the providers), percentage thinning is used so that the probability of buying a certain type of resource is never higher than the number of remaining tokens over the number of expected needs during the reminder of the interval. For more details we refer to Arvidsson et al. [1].

The Hierarchical Auction-Based Architecture One possible implementation of a distributed, synchronous system is the *hierarchical auction* (HA) architecture [11]. The idea is to partition the set of allocators and to use one distributor for aggregating bids and holding auctions for each partition. These distributors then connect to higher order distributors in a hierarchical manner until the total demand can be matched against the amount of available resources offered by the quantifiers.

The Centralized Leaky Bucket Architecture The centralized asynchronous architecture we have chosen is based upon an asynchronous approach called *Leaky bucket* [2]. The basic idea is that each provider is equipped with a Leaky bucket that feeds requests to the provider at an even and optimal rate. This is done by inserting the incoming requests from the customers in a queue in the Leaky bucket. These requests are then dequeued at a rate corresponding to the maximum capacity of that provider. If the queue is full, the requests are rejected.

To get a *centralized* architecture, we introduce a *centralized leaky bucket (*CLB*)* architecture, in which there is just one central distributor, common for all allocators and quantifiers. The allocators send all requests immediately to this distributor, which consists of a common leaky bucket for queuing the requests. It also has a router that continuously dequeues requests at a rate corresponding to the total capacity of the providers and then forwards the requests evenly to the providers in proportion to their capacity. If the bucket is full, the request is returned to the allocator where it is rejected.

The Mobile Broker Architecture As an example of a distributed, asynchronous system, we choose a *mobile broker* (MB) architecture [4]. In this architecture, the distributors are implemented as mobile brokers (one for each provider) that sequentially visit each (or a subset) of the allocators offering the resources currently available at the corresponding provider The allocator then *requests* the resources it needs for the moment (or rather, predicts it will need in the near future). If possible, the broker gives this amount of resources to the allocator. Otherwise, it gives as much as is currently available at the provider. However, there are two problems with this naive approach:

- If an allocator demands all the available resources, the broker will give them to that allocator. Thus, the broker will not be able to hand out any more resources for a while, which would not be fair.
- If the overall load is low or moderate, the allocators are given just as much resources as they demand. However, if an allocator need slightly more resources than it asked for (predicted), it will have to turn down request, even though the provider has lots of surplus capacity.

In order to solve these problems, we use a broker mechanism that strive to give out all the available resources and give each allocator resources in proportion to their part of the total current demand (of the allocators in the route). For the details of this approach we refer to Johansson et al. [7]. It should be noted that in case of a sudden increase in demand, the resources given out may momentarily exceed the available resources, which in the worst case will lead to a transient overload situation. However, the mechanism is self-stabilizing, and will find an equilibrium within one route (given that the demands are relatively stable). The mechanism ensures that the allocators are given resources that (relatively) correspond to their share of the total demand handled by the broker, thus solving both problems in the naive solution above.

If an allocator are visited by several brokers it may happen that some of the brokers' SCP are carrying a higher load than the others. To deal with this problem an additional balancing function is used, making the allocators try to move load from those SCPs with relatively high load to those with relatively low load. The allocator calculates the load of a broker from the quotient between what it asked for and what is was given by the broker.

3.2 Concrete Attributes

We now operationalize the abstract attributes presented earlier. Thus, in the domain of IN load management the attributes are defined as follows:

- Reactivity is measured by how fast the MAS is able to re-allocate the available SCP processing time when there are sudden changes of offered loads by the SSPs.
- Load balancing is measured by the standard deviation between the carried load of the SCPs.
- Fairness is measured by the standard deviation of rejected calls divided by the accepted calls between the SSPs, i.e., the rejection rate.
- The utilization of resources is measured by how close the carried load is to the target load, or offered load, if the offered load is less than the target load. SCP load levels should be as close to the target load (e.g., 0.9 Erlangs, corresponding to 90% of its capacity) as possible but not exceed it. If an overload situation is approaching, the SSPs should throttle new requests.
- Responsiveness is measured by the time it takes for the SSPs to get response from an SCP.
- Communication overhead is measured by the bandwidth necessary for the MAS to perform the reallocation.
- Robustness is measured in terms of the consequences of a distributor agent failure.
- Modifiability is measured in terms of how easy it is to add a new or remove an existing SSP or SCP.
- Scalability is measured in terms of how the number of SSPs and SCPs influence the performance-related attributes.

3.3 Experimental Evaluation

The four concrete architectures have been evaluated in simulation experiments. For these experiments we used the same simulation model as Arvidsson et al. [1]. The Intelligent Network modeled has 8 SCPs and 32 SSPs, which are assumed to be statistically identical respectively. The processors at the SCPs are engineered to operate at 90 percent of the maximum capacity (target load is 0.9 Erlang). All messages passing through the network experience a constant delay of 5 ms, and it is assumed that no messages are lost. Detailed descriptions the simulation results can be found in [8]. Since we use these experiments as a case study, we only summarize the results here:

- As the theoretical evaluation predicted, reactivity was promoted by the two asynchronous architectures. For instance, we simulated a scenario in which half of the 32 SSPs experience an offered load corresponding to 0.2 Erlang, and the other half a load corresponding to 1.4 Erlang. The whole system thus is offered a total average load of 0.8 Erlang, which is below the target load, and therefore possible for the system to carry. At time 400, the levels of loads are shifted from high to low and vice versa and 10 seconds later they are swapped back again, see Fig. 2. As can be seen in Fig. 3, the CLB manage this difficult situation perfectly and the other asynchronous architecture MB does almost as well. The synchronous auction-based architectures, on the other hand, have apparent difficulties to adapt to the changes.

– All the architectures were able to balance the load well. There were just small differences in the standard deviations (of the measured carried load of the SCPs), varying between 1 and 3 mErlang depending on the offered load. When the offered load was below target load the centralized architectures performed slightly better, whereas there were no measurable differences in overload situations.
– With respect to fairness, all architectures performed well when the offered load was below target load. In overload situations, MB was significantly less fair than the other architectures. However, it may be the case that an improvement of the design of the broker routes may reduce these differences.
– It was not clear from theoretical analysis if there is any correlation between the ability to utilize the resources and the architectural properties. The simulation results presented to the left in Fig. 4 indicates that the centralized asynchronous architecture performs best in this respect.
– Also, it was not clear the theoretical analysis if there is any correlation between responsiveness and the architecture properties. But in this case the simulation results indicate that the centralized asynchronous architecture has the worst performance. See Fig. 5. Note that responsiveness is measured in terms of the average delay for the individual requests, whereas reactivity is a measure of the delay of reallocations which typically concerns a large number of requests.
– We tuned the parameters in the simulation experiments so that both CA, HA and MB needed approximately the same bandwidth for communication overhead, about 40 messages/second irrespective of the amount of offered load. The bandwidth needed by CLB, however, is proportional to the number of requests and is considerably higher than for the other architectures. For instance, when the offered load is 0.70 Erlang, 2150 messages/second are sent, and when the offered load is 2.0 Erlang, 9365, i.e., more than 200 times as much bandwidth as the other architectures need.

For the general software architecture attributes (robustness, modifiability, and scalability), we refer to the theoretical analysis in section 2.3.

4 Conclusions and Future Work

We have described a systematic way of evaluating different aspects of different MAS architectures. This approach was applied to an abstract domain, namely dynamic resource allocation, and a theoretical evaluation of abstract architectures was made. This was supplemented by an experimental study of an implementation of this abstract domain, load balancing and overload control in Intelligent Networks. The experimental evaluation confirmed the conclusions of the theoretical analysis, e.g., that asynchronous architectures are able to react faster than synchronous. In addition, it gave insights concerning attributes for which no clear conclusions could be achieved from the theoretical analysis, e.g., that centralized asynchronous architectures are able to utilize the available resources better than

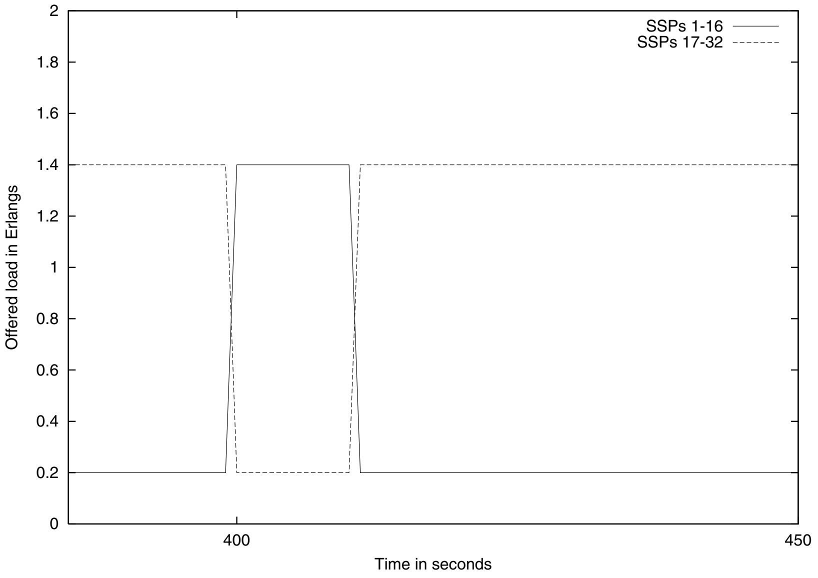


Fig. 2. Reactivity: The expected offered load

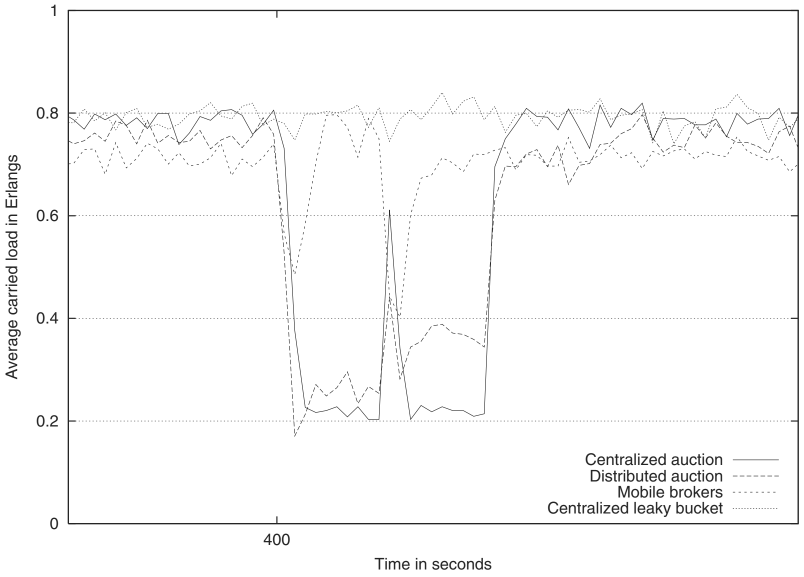


Fig. 3. Reactivity: The average carried load

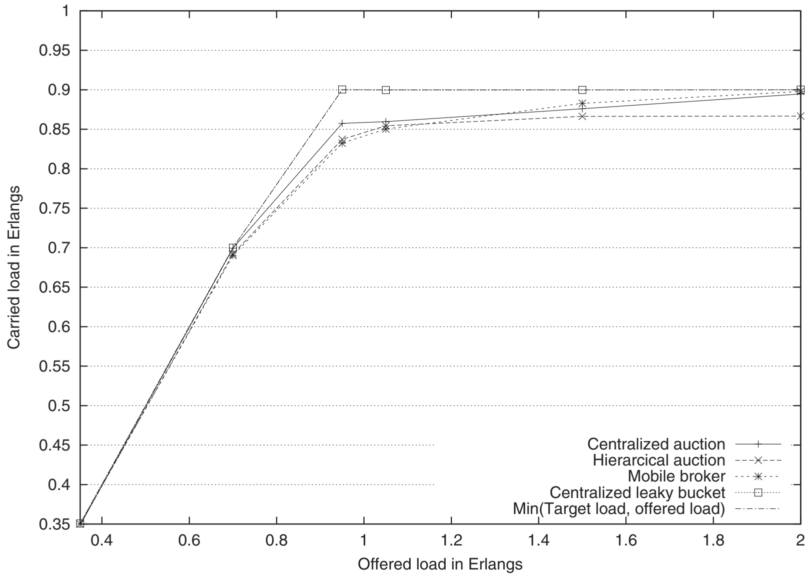


Fig. 4. Utilization of resources: The ability to carry different levels of offered load

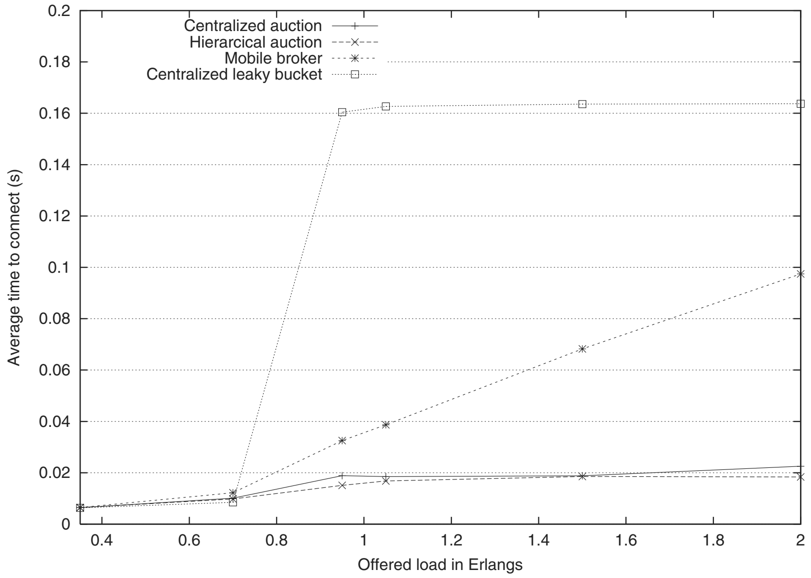


Fig. 5. Responsiveness: The average response time for different levels of offered load

the other architectures, but have larger delays and need more bandwidth when the load is high.

The results of the case study were, not very surprisingly, that different architectures excel in different dimensions. The choice of MAS architecture for a particular application should be guided by the balance of the trade-off between these dimensions that is optimal for that application. We believe that if the systematic approach suggested here is widely adopted, such choices can be more informed than is currently practice.

Our plans for future work includes:

- Further experimental validation in the IN domain of the theoretical results regarding, e.g., scalability.
- Experimental validation in another dynamic resource allocation domain.
- Use the outlined approach to investigate other domains than dynamic resource allocation.
- Investigating to what extent the implementations of the individual agents influence system performance.
- Develop a more refined method for characterizing MAS architectures.

Acknowledgments

The authors would like to thank Martin Kristell for assistance during the simulation experiments and the Swedish Knowledge Foundation for funding parts of this work.

References

[1] Å. Arvidsson, B. Jennings, L. Angelin, and M. Svensson. On the use of agent technology for IN load control. In *Proceedings of the 16th International Teletraffic Congress*. Elsevier Science, 1999. 176, 178

[2] A.W. Berger. Comparison of call gapping and percent blocking for overload control in distributed switching systems and telecommunication networks. *IEEE Trans. Commun.*, 39:574–580, 1991. 176

[3] E. Bodanese and L. Cuthbert. An intelligent channel allocation scheme for mobile networks: An application of agent technology. In *Proceedings of the 2nd International Conference on Intelligent Agent Technology*, pages 322–333. World Scientific Press, 2001. 172

[4] B. Carlsson, P. Davidsson, S. J. Johansson, and M. Ohlin. Using mobile agents for IN load control. In *Proceedings of Intelligent Networks '2000*. IEEE, 2000. 177

[5] S.H. Clearwater, editor. *Market-Based Control: Some early lessons*. World Scientific, 1996. 176

[6] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. Addison Wesley, 2002. 173

[7] S.J. Johansson, P. Davidsson, and B. Carlsson. Coordination models for dynamic resource allocation. In A. Porto and G.-C. Roman, editors, *Coordination Languages and Models*, volume 1906 of *Lecture notes in computer science*, pages 182–197. Springer Verlag, 2000. Proceedings of the 4th International Conference on Coordination. 177

[8] S. J. Johansson, P. Davidsson, and M. Kristell. Four architectures for dynamic resource allocation. In *Proceedings of Fourth International Workshop on Mobile Agents for Telecommunication Applications*, 2002. Springer Verlag, LNCS. 178

[9] P. B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, July 1995. 171

[10] T. Magedanz and R. Popescu-Zeletin. *Intelligent Networks*. International Thomson Computer Press, 1996. 174

[11] F. Ygge. *Market-Oriented Programming and its Application to Power Load Management*. PhD thesis, Lund University, Sweden, 1998. 172, 176

Engineering Agent Systems for Decision Support*

Sascha Ossowski[1], Josefa Z. Hern· ndez[2],
Carlos A. Iglesias[3], and Alberto Fern· ndez[1]

[1]AI Group, University Rey Juan Carlos
Campus de MÛstoles s/n, E-28933 Madrid
{s.ossowski,al.fernandez}@escet.urjc.es
[2]AI Dpt., Tech. University of Madrid
Campus de Montegancedo s/n, E-28660 Madrid
phernan@dia.fi.upm.es
[3] Technical Innovation Dpt.
Germinus XXI, Gran VÌa 1 - 2∫ Izq, E-28013 Madrid
cif@germinus.com

Abstract. This paper discusses how agent technology can be applied to the design of advanced Information Systems for Decision Support. In particular, it describes the different steps and models that are necessary to engineer Decision Support Systems based on a multiagent architecture. The approach is illustrated by a case study in the traffic management domain.

1 Introduction

Decision Support Systems (DSS) are information systems that provide assistance to humans involved in complex decision-making processes. Early DSS were conceived as simple databases for storage and recovery of decision relevant information [19]. Despite some intends to improve organisation and presentation of such data by means of additional deductive facilities, it soon became apparent that the key problem for a decision-maker is not such much to *access* pertinent data but rather to *understand* its significance. So, the fundamental task for modern DSS is to help decision-makers in building up and exploring the implications of their judgements, so as to take decisions based on understanding [6].

DSS are particularly relevant in domains where human operators have to take decisions regarding the management of complex industrial or environmental processes: controlling road traffic flows [7], managing dams in a watershed basin [8], or administering large computer networks [20]. The increasing data volume and the decreasing time horizon within which control decisions have to be taken, have generated a need for DSS; they evaluate data about the system state, collected either directly or through

* Work supported by the Spanish Ministry of Science and Technology (MCyT) under grant TIC2000-1370-C04.

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 184-198, 2003.

real-time databases, so as to warn operators of any undesired evolution and to answer their questions concerning potential reasons, effects and countermeasures [4].

Recently, the concept of *Intelligent DEcision-making Assistants* (IDEA) has been proposed [18]. These are intelligent agents that render support to their human operators in the various stages of their decision-making processes by means of flexible dialogues. Suppose a DSS that assists operators in a traffic control centre to generate and choose among alternative signal plans for traffic control devices (Variable Message Panels, Traffic lights etc.), so as to assure a smooth flow of traffic, as well as to avoid and/or overcome potentially critical situations. It is important for such a system to support a *reactive* mode of interaction. Once an operator detects some abnormal system parameters or receives alarm messages (e.g. from traffic observers), she initiates a dialogue with the DSS in order to prepare her decision. There are several questions that she will put forward to the agent, e.g. respecting the cause of the current situation (*What is happening*?), the reason for it (*Why is it happening*?), as well as action alternatives (*What can be done*?) and their potential effect of different alternatives (*What may happen if* ?). These questions will trigger several processes in the course of which the agent autonomously gathers relevant data from the different information sources available. On the basis of this information, it will apply its domain knowledge to generate answers to the questions, taking into account rationality constraints with respect to the operator's objective of minimising negative impacts on the part of the road network that she is responsible for. Another scenario is given when the intelligent assistant monitoring the road network detects symptoms of deterioration of the traffic situation or of dysfunctional active signal plans (initiated, or not adequately updated, for instance, by a less trained or experienced operator). In this case, it is to *proactively* issue warnings, initiating a dialogue with the decision-maker, in the course of which the latter explores the reasons and implications of that warning in a flexible and intuitive manner.

This paper discusses how agent technology can be applied to the design of advanced DSS. In particular, setting out from our past experience in this field, it describes the different steps and models that are necessary to engineer intelligent multi-agent systems for Decision Support. It is organised as follows: Section 2 describes how agent-based methodologies can be used for a principled design of IDEAs. Section 3 presents a case study respecting analysis, design, implementation and operation of such systems. We conclude this paper pointing to present and future lines of research.

2 Engineering IDEAs

Current research in Agent Oriented Software Engineering examines principled ways for constructing complex software systems based on the agent metaphor [10],[12],[21]. Often, traditional *object-oriented methodologies* such as UML [11] are extended to account for the characteristics of agent systems, paying particular attention to the modelling of agent interactions and agent architecture. Such methodologies also appear to be an adequate choice for the construction of IDEAs, but need to be complemented by techniques from the *knowledge engineering* field, as decision support agents draw heavily upon complex, knowledge-based reasoning processes. To this respect, the well-known *CommonKADS* approach [1], that supports advanced knowledge modelling, is particularly relevant.

In the sequel we describe how to apply agent-oriented software engineering techniques from both fields in order to engineer intelligent assistants for decision support. First, we use the UER modelling technique, an extension of UML, for analysis. Then, a *CommonKADS*-like approach is used for the identification of basic reasoning tasks of IDEAs. Finally, we show how to obtain and structure reasoning methods based on such a knowledge-oriented approach [5].

2.1 Analysis

In this section we describe how to obtain a conceptualisation of IDEAs on the basis of the UER (User-Environment-Responsibility) technique [9]. It is particularly convenient for our purposes, as it allows for both, monolithic decision support agents as well as for IDEAs that are themselves based on a multiagent architecture which, according to our experiences, will often be the case. The UER technique analyses the system from three different perspectives:

- *User-Centred Analysis*. The potential users (called actors) of the agent system are identified, together with their possible tasks or functions. The result of this analysis is the set of use cases. This analysis answers the question: What are the possible uses of the multiagent system?
- *Environment-centred Analysis*. Agents are situated in an environment, and this environment needs to be modelled. In particular, we are interested in modelling how the system can act and react to this environment. The result of this analysis is the set of reaction cases. This analysis answers the question: How does the agent system react to the environment?
- *Responsibility-driven Analysis*. In contrast with usual software systems, agent systems can act proactively. The user can desire that the system has some responsibilities, that is, the user can assign some goals or responsibilities to the system and the system carries out these responsibilities without a direct demand. This analysis answers the question: What are the goals of the system? The main difference between *goal cases* and use cases, is that the latter show how the system gives an answer to a user request, while the former show how the system behaves when some condition is fulfilled.

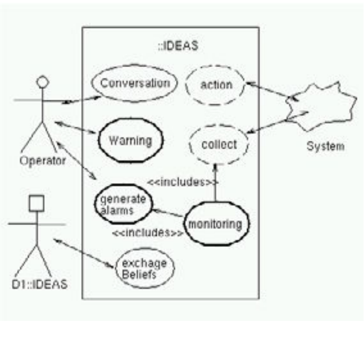


Fig. 1. UER Model of IDEAs

UER defines different UML stereotypes (see Fig. 1) for every modelling element: use cases are UML standard ellipses, goal cases are ellipses with wider lines, reactive cases are ellipses with discontinued lines, environment objects have an irregular form, human actors are standard UML actors and software actors are square headed actors). In the sequel, we model a generic Intelligent Decision Support Agent System in these terms. Fig. 1 summarises the results.

Not surprisingly, in the general case, user-centred analysis identifies at least two actors: the decision-maker and the decision support system, i.e. the IDEA. The most relevant generic use cases that these actors are involved in include:

- *Conversation*: The user requests some explanation from the system.
- *Exchange-beliefs*: another DSS interchanges plans with the systems to get a broader understanding of the situation.

The use case conversation can be detailed as:

- *Why*: the decision-maker requests an explanation of information provided by the DSS.
- *What*: the decision-maker requests relevant data to take a decision.
- *What-if*: the decision-maker asks to evaluate the results of a decision and its consequences.
- *Suggestions*: the decision-maker requests possible actions to be taken.

Depending on the particular domain about which the IDEA is knowledgeable, the *environment-centred analysis* identifies relevant environment objects, and every possible event generated from these objects and possible actions carried out on them. For example, in a financial DSS, we could identify an object *share*. The possible input events are new values of this share, and the possible actions to buy or to sell.

Finally, in the *goal-driven analysis* we need to identify the autonomous behaviours of an IDeA. Its most important responsibilities usually are:

- *Monitoring*: observe the environment and detect problematic behaviours;
- *Alarm generation*: raise alarms if there is a critical situation;
- *Warning*: warning respecting undesired consequences of ìbadî actions and potentially suggesting better ones.

2.2 Task Design

In methodologies that go back to the knowledge engineering field, a task is usually conceived as an abstract description of how the world (or an agentís ìmental modelî of it) needs to be transformed in order to achieve a desired behaviour or functionality. In this section we identify the different tasks that an IDEA needs to cope with so as to be able to behave successfully in the aforementioned conversation cases.

To generate answers for the different classes of questions in our conversation framework, we have identified essentially four tasks.

- *Problem identification.* From the analysis of the information received from a communication infrastructure or directly from the operator, an IDEA classifies the state of the monitored system.
- *Diagnosis*. The presence of unacceptable events or situations requires an explanation in terms of causal features of the situation.
- *Action planning*. Once a problem has been identified, a possible sequence of actions applicable on the causes may be established.
- *Prediction*. the consequences of events and operator actions are simulated.

Suppose some system components *S*, some external events *E* and operator actions *A*. By combining the above tasks in different manners, several questions that a decision-maker typically faces can be answered. For instance:

- ìWhat is happening in S ?î : problem identification + diagnosis.
  A diagnosis *D* for some potential malfunction is produced.
- ìWhat to do on D in S ?î : action planning + prediction
  Decision options are shaped and their potential effects evaluated.
- ìWhat may happen if *E* in *S* ?î : *prediction* + *problem identification* + *diagnosis* Potential future problems in evolution of the system are identified.
- ìWhat to do if E in S ?î : prediction + problem ident. + diagnosis + planning

Decision options respecting potential future problems are outlined.

Still, in complex domains, like the ones that DSS are usually being applied to, the system designer often has to deal with different types of *a priori* distribution [17], which makes it unfeasible to cope directly with these tasks. This distribution may be implied by physical requirements, as many environments show a natural spatial distribution. DSS that assist decision-makers in managing the spill gates of several dams in a watershed basin, for instance, rely on data on weather conditions and water levels from sensors that are geographically distributed. But distribution may also be implied by organisational requirements. To be successful, a DSS aimed at helping managers to maintain a smooth flow of work in a company will have to respect the context of its existing human organisation, e.g. its *a priori* distribution of responsibilities and tasks. Another source of distribution is the availability of knowledge. If, as in the road traffic management domain, a DSS relies on the knowledge elicited from human experts, and by experience these experts conceive traffic behaviour in terms of certain problem areas, then the system will have to reflect this a priori distribution.

A common way of dealing with these issues it to conceive an IDEA itself as a *multiagent* system, where each distributed entity is controlled by an agent. By consequence, any of the aforementioned tasks of problem identification, diagnosis, action planning and prediction can be performed locally by each agent within the multiagent system that makes up an IDeA. These local tasks will be of less complexity, but they are also interdependent: the cause of a rising water level at a certain dam may be the opening of a spill gate further upstream, the predicted completion of a work process in a company will depend on the timeliness of any of its work cells, and the effectiveness of action plan in road traffic management will rely on a globally consistent use of control devices. The co-ordination task, that such a multiagent system faces, refers to the management of these dependencies between local tasks.

2.3 Method Design

Most knowledge-oriented methodologies make use of the concept of problem-solving methods in order to cope with tasks. In particular, such methods indicate *how* a task is achieved, by describing the different steps by which its inputs are transformed into its outputs. The problem-solving process associated to a task is structured: each of its steps may set up several subtasks, which again are to be solved by simpler methods etc, until some elementary tasks can be achieved directly. In the sequel, we identify different such ìreasoning skeletonsî that our IDEAs will need to apply so as to cope effectively with the tasks identified in the previous section.

Problem Identification Methods

A classification method with two options may be applied:

- Identification of a reference situation and classification of the differences between the reference and the current situation.
- Direct classification of the current situation based on a predefined taxonomy where problems of different types are described.

The first approach requires: (1) to infer from the current situation the evolution of parameters consistent with the functional and structural constraints which optimises a collection of predefined criteria (e.g. in a given congested situation an ideal assignment of traffic flows could be identified, adapted to the available capacity of the network and to the traffic demand between entries and exits to the network), and (2) to classify the differences between the observed situation and the resulting class of situations according to a hierarchy similar to the one previously commented.

The first approach, then, applies a method in two steps. The first step derives a possible new state from the current situation that may be supported by an *ad hoc* procedure, adapted to the characteristics of the domain model. For the second subtask a primary representation based on rules and/or frames may be applied in a hierarchical *establish & refine* model [3]. The second approach is similar to the first one, but in this case a complete description of the situation is required, not only the differences with the reference situation.

Diagnosis Methods

This task infers a collection of causes explaining the problems identified by the previous one. Several methods may be directly applied: (1) The *classification* method, which extends problem type frames by additional cause attributes in such a way that once a problem pattern has been selected, the cause features assumed for this problem type are assumed. (2) A version of the *cover & differentiate* method [16] where a hierarchical approach to an explanatory set of causes is generated through the following reasoning steps: (i) from the attributes of the type of problem detected a collection of possible causes may be inferred *covering* these values (i.e. if the causes inferred are true the problem feature values are also true), (ii) since this first set of causes may be too large, a deeper analysis to *differentiate* subsets explanatory enough is necessary. To do this, knowledge about the individual impact of sets of causes should be used to decide which causes may be erased. To obtain those enlarged impact estimates knowledge relating cause sets and impacts must be available in terms of relations or in terms of simulator.

A hierarchy of conjunctive cause sets where the resulting impacts are known may be established from aggregated sets to more disaggregated ones, every node with the condition that there is a method to evaluate the resulting effects of the causes included in the node.

This second analysis may be done in several steps in such a way that several reasonable partitions in subsets of interacting causes are selected in the hierarchy as potentially explanatory. It may be followed by a step of selection of the more efficient subset (i.e. which explains better the problem features) that may be also partitioned until a minimal explanatory set is found in the hierarchy. For instance, in the domain of traffic it may be identified a problem with three potential congested critical sections, the initial candidate causes are sets of paths between entry and exit points in the area network passing through the congested points. The process of differentiation aims to identify, at the different points, the participation in the traffic excess of a subset of those paths which provide the significant part of this excess and hence, the paths where the traffic flow must be withdrawn using control devices (messages or traffic lights) to reduce the congested demand.

Action Planning Methods

After the problem identification and diagnosis tasks, some scenarios of causes of problems have been deduced together with its impacts. The *action planning* task must generate a consistent set of actions oriented toward the reduction or elimination of causes and/or toward the reduction of impact damages where no possible cause reduction may be produced.

Specifying this task in a general way requires defining the elementary actions that will be the basis for definition of acceptable decision plans together with their models. The action planning task may be performed by a method integrating consistent sequences of actions to transit to a situation where most damages have been alleviated and all the problem causes have been cancelled.

The general reasoning method to deal with this functionality is a planner. However, it is assumed that the area of expertise to be modelled embodies structured knowledge criteria that are sufficiently precise to generate a plan in a more simple way, using classification reasoning on predefined plans and subplans. According to this assumption, it is reasonable to apply a stepwise reasoning method derived from the routine design method proposed by [2].

The declarative knowledge may be organised with collections of plans capable to perform subtasks that may be integrated to build plans aiming to perform more complex tasks. A plan may be defined by sequences of elementary actions or may include other subtasks supported by other collections of plans. The method using this domain knowledge is a type of *skeletal plan refinement* supporting a progressive refinement of the subtasks from intermediate partial versions of the plan in the following steps:

- A basic method M is applied which introduces a collection of consistent actions together with a decomposition of the general task in plans of actions a_i integrated with some subtasks not yet formulated as plans $< a_i , a_j ,..., T_l , T_m ,..., a_s >$ where T_l, T_m are subtasks with no plan of elementary actions specified.

- For every subtask not yet described in terms of basic actions $(a_i,\ a_j,...)$ such as T_l or T_m an inference step of the same type is applied by using an instantiation of *M* with other premises and domain knowledge. A tree exploring different steps of plan detail is produced until detailed plans are obtained or an application of *M* fails and a backtracking is produced to alternative task decompositions.

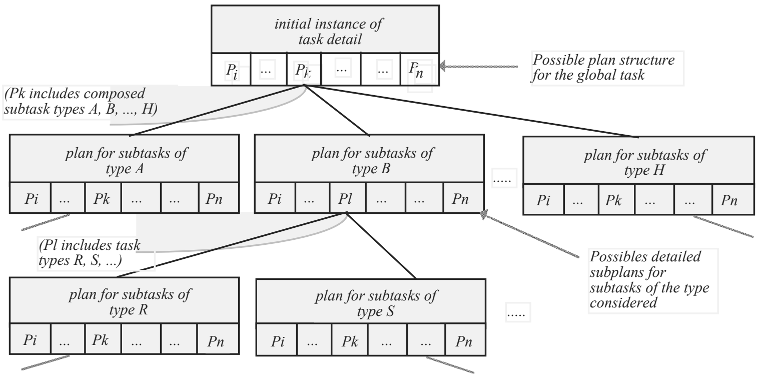

initial instance of task detail
Pi
...
Pk
...
...
Pn
Possible plan structure for the global task
(Pk includes composed subtask types A, B, ..., H)
plan for subtasks of type A
Pi
...
Pk
...
...
Pn
plan for subtasks of type B
Pi
...
Pl
...
...
Pn
.....
plan for subtasks of type H
Pi
...
Pk
...
...
Pn
(Pl includes task types R, S, ...)
Possibles detailed subplans for subtasks of the type considered
plan for subtasks of type R
Pi
...
Pk
...
...
Pn
plan for subtasks of type S
Pi
...
Pk
...
...
Pn
.....

Fig. 2. The general plan refinement strategy

The general search process of the method *M* is summarised in Fig. 2 where every node box symbolises the application of *M* with different declarative knowledge blocks of the four types commented before. The interest in using this reasoning strategy is that it may embed the simple case where only a classification step is considered when a collection of totally defined plans (i.e. with no T_l or T_m undefined tasks) is given in the domain model. In this case, the general reasoning strategy will stop after the first step once some task detail modules are applied proposing several complete plan options.

Behaviour Prediction Methods

This task has as main goal to propose scenarios of short-term future behaviour of the different components of the model. There may be specific simulation methods performing this type of task. A library could be considered to support a class of applications including a collection of typical physical components. However, it could be considered a simplified model to perform short term general *black box* behaviour simulation task, by using an envisionment graph formed by classes of component situations, described every one by a collection of slot values corresponding to every state attribute, and transition links representing types of external actions or decisions producing the change from one state to the other. In simple systems this type of representations may be useful. They may be generated through the abstraction of the results obtained from previous numerical model application. It must not be forgotten that the level of precision required for these models is not very high. It should be enough to differentiate the possible types of short-term evolution of the controlled system. The model of reasoning may take the current state from the information system and the assumptions about the external actions and match it with some node in the graph. As a result, for every matched situation the predictable short-term changes are described by the downstream connected states.

Co-ordination Methods

Co-ordination is best conceived of as the management of dependencies between activities [15,17]. Methods that perform this type of management usually comprise three steps:

- *Dependency detection*: using domain knowledge about the different dependencies that may occur (producer-consumer relationships, resource limitations etc. [15]) positive and negative relationships between the different local tasks of the agents are detected. For instance, two local traffic agents may want to display different warning messages on the same traffic message panel.
- *Option generation*: for every dependency, the set of possible management actions is generated. In our traffic example, any of the involved agents may change its local action plan, or we may merge the incompatible messages ìincident at *A*î and ìcongestion at *B*î to be displayed on the same panel into the message ìtraffic problems at *A* and *B*î.
- *Management decision*: finally, a decision must be taken respecting the dependency management action to be applied. In the traffic example, one possible criterion for taking this decision is the aim of distributing traffic load equally among the different problem areas.

3 An Example: IDEAs for Traffic Control

We now illustrate the above framework for engineering Intelligent Decision-making Assistants by an example. We will describe the construction of IDEAS for road traffic management, a real-world domain, and an example of the high degree of complexity that decision support applications have to deal with. For this purpose, we set on from the TRYS family of system (e.g. [5,7,17]), agent-based DSS that have been build and used experimentally in different Spanish towns. In this section, we provide a principled ìredesignî of the architecture of these systems, along the lines proposed in the previous section. First, our particular traffic management domain is sketched and the requirements for traffic management IDEAs are analysed in terms of an UER model. Then, the design of such IDEAs from a knowledge-oriented point of view is described, so as to finally sketch some implementation issues.

3.1 The Traffic Problem: Analysis

In Barcelona, the local traffic control centre JPT is in charge of managing urban road transport, so as to maintain and restore the ìsmoothî flow of vehicles. Traffic engineers continuously receive information about the traffic state, identify potential problems, and act upon control devices to overcome them. It has become particularly difficult for the JPT engineers to perform this job in real time, as in the follow-up of the 1992 Olympic Games the traffic management infrastructure in Barcelona has become increasingly complex. Nowadays, information about the traffic state of the urban motorway network, consisting of one ring road and seven adjacent motorways, is provided by over 300 telemetered sensors (ìloop detectorsî) via fibre optics communication links. Control actions can be taken by means of 52 Variable Message Signals (VMS), 3 traffic lights for junction control, as well as by ramp metering on 7 ring-road drives. Fig. 3 illustrates typical elements of this traffic management infrastructure.

JPT traffic controllers logically subdivide the road network into *problem areas*, for which they are able to generate efficient signal plans. Still, problem areas overlap, so potential conflicts between local signal plans need to be taken into account in order to obtain globally consistent signal plans..

Fig. 4 outlines our model of an IDEA-based DSS for this domain. The ultimate goal of the traffic decision support system is to assist human operators in the management decision by increasing their awareness of the traffic situation and the options to influence it. For the purposes of this article, we consider only a human actor, the operator. A complete analysis might consider additional actors such as administrator of the system or different levels of operators. The main use cases of an operator are:

- *EnterCaseFromSensors*: the operator receives data from the sensors from an external system and he/she introduces the new case manually.
- *EnterAlarmFromPolice*: the operator receives an alarm from the police.

Signal plans generated by operators comprise two classes of actions:

- *PutAMsgOnPanel*: set a warning message on a panel (e.g. ìcongestion at *X*î)
- *ModifyRegulators*: modify a regulator in order to prevent congestion

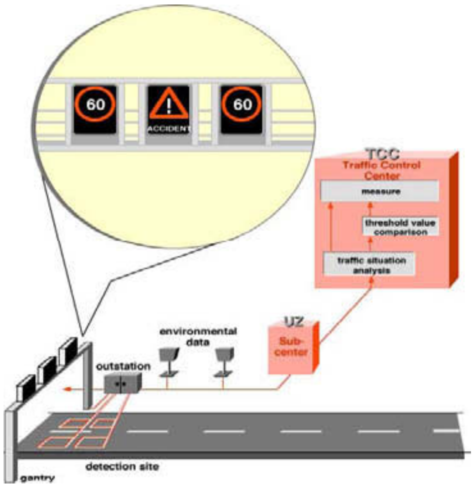


Fig. 3. Traffic infrastructure ([13])

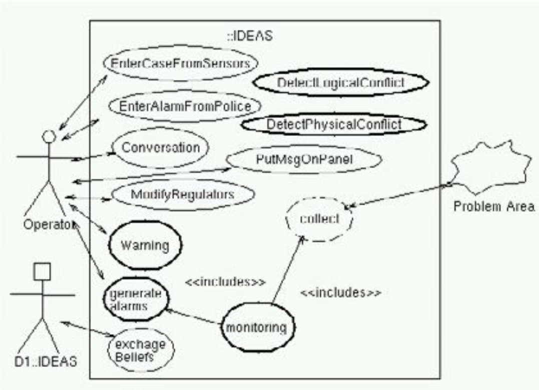


Fig. 4. UER Diagram for the traffic domain

When communicating with the DSS, the operator is involved in flexible dialogues, asking for the reason of a diagnosis (*why*), what is happening (*what*) or a specific simulation (*what-if*). The following goal cases are assigned to the DSS:

- *DetectPhysicalConflict*: detect when two actions from adjacent areas generate different actions over the same area elements.
- *DetectLogicalConflict*: detect if actions from adjacent areas generate a bad overall solution.

The DSS can modify an environment problem (*publishAPanel*, *switchRegulator*). There are no current requirements for real reactivity, but one might consider reactive extensions like speed limitations in case a severe incident is detected. Once the UER cases have been defined, a first identification of the agents can be done. In this case, since we had distributed expertise in the problem, each one agent has been assigned each problem area of the road network.

3.2 IDEAS for Traffic Management: Design

The magnitude of a traffic problem in certain part of the road network can be expressed by the amount of traffic demand that exceeds the capacity of a certain road segment (in vehicles per hour). This is called the segmentís *traffic excess*. The quality of a traffic management action can be measured by the reduction of traffic excess, that they are expected to produce. In consequence, traffic management agents use the overall reduction of excess in their problem areas (i.e. the sum over all road segments that belong to their area) as a local utility measure. Setting out from this notion, in the sequel we show how to design methods that cope with an IDEAís tasks.

Problem Identification and Diagnosis

Every couple of minutes, a traffic management agent receives temporal series of magnitudes such as traffic speed, flow and occupancy from the road sensors of its area. This raw data is initially pre-processed in order to filter out noisy and erroneous data. Subsequently, fuzzy data abstraction is performed (see Fig. 5), and aggregate magnitudes such as temporal and spatial gradients are calculated for the different sections.

The actual problem identification is performed by matching the abstracted traffic data against a knowledge base of frames, which model problem scenarios. Fig. 6 shows one such frame that matches the abstracted traffic data. Suppose that as a result of data abstraction low speed and high occupancy are identified in Ronda de Dalt en Diagonal and medium to high speed and low occupancy in Ronda en díEslugues. These facts match the frame shown in Fig. 6, so that an incident in the central lane of Diagonal road is identified, which manifests itself as a traffic excess (with respect to the roadís capacity) of 2200 veh/h between Diagonal and Llobregat in the Dalt ring-road. Traffic from Collcerola to Llobregat and, in a minor degree, from Diagonal heading towards Llobregat contributes to this excess.

Action Planning and Prediction Tasks

The action planning and prediction tasks adhere to the following line of reasoning: first, the historic traffic demand between nodes is retrieved and the contribution of each path to the problem in the critical section calculated. This is done by matching the current abstract traffic state and the state of the control devices against a knowledge base of frames, representing traffic distribution scenarios.

Finally, coherent alternative signal plans are generated by using the distribution scenario frames once again: every frame applicable to the current situation is preselected. Assume that this is the case for the frame shown in Fig. 6. Its short-term effects are estimated by simulating its impact on the current traffic situation. This is done by using network structure knowledge to assign traffic demand to the road network, in accordance with the distribution of traffic volume among paths that the frame specifies. In the example about one half of the traffic volume from Collcerola to Llobregat will pass through the Dalt ring-road, while a smaller amount chooses a path through Can Caralleu or other alternative paths, if the corresponding signal plan is set. If the simulation shows a reasonable decrease of excess in the critical section, the frameís signal plan constitutes one recommendation of the system. In the example, it is suggested to display congestion warnings at Diagonal for panels 17PIV1, 13PIV2 and 8PIV1, while setting the contention level of regulator R1 to medium. As a result of this process, a set of alternative signal plan recommendations, together with their utility (i.e. their expected reduction of local traffic excess) is produced.

Co-ordination

Methods for co-ordinating the different traffic management agents may be either centralised or decentralised, leading to different architectures for traffic management IDEAs. The InTRYS system [5] relies on a distinguished co-ordinator agent to perform co-ordination. Local traffic management agents send their control plans to that agent. Dependency detection and option generation relies on a knowledge base of rules. The dependency management decision is taken on the basis of priority knowledge, indicating which problem area is most critical, and thus determining which traffic agent needs to revise its control proposal. By contrast, the TRYSA$_2$ system [17] relies on a decentralised co-ordination mechanism. Dependency detection and option generation is done in a similar fashion as in the InTRYS system, although the corresponding knowledge bases are distributed and contain only the information necessary to deal with dependencies among neighbouring agents. Still, the management decision emerges from a negotiation among agents, which is based on a game-theoretic framework. The traffic management agents are supposed to be self-interested and mutually ìthreatenî each other with the potential consequences of failing to reach on agreement (although this never happens, as in this domain agents are always better off if the cooperate). The compromise, that they finally converge on, is a reflection of the relevance of an agent in a particular situation. When tuning the system, the designer can bias this basic compromise in a desired direction be issuing certain ìprescriptionsî on the use of resource by agents [17].

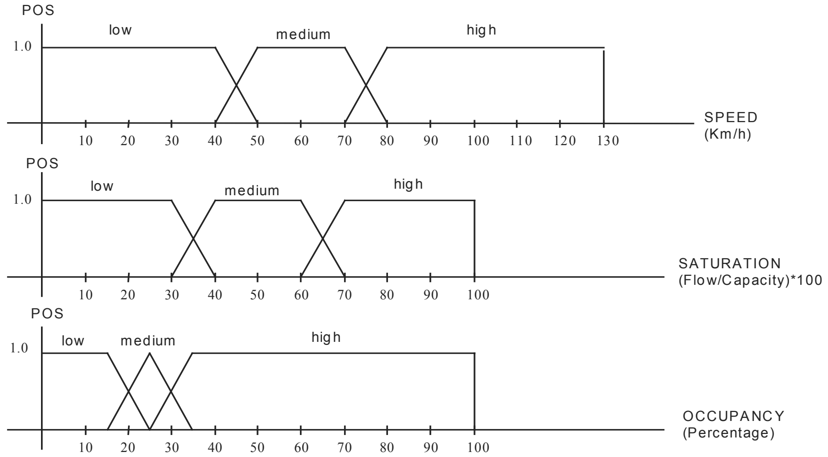
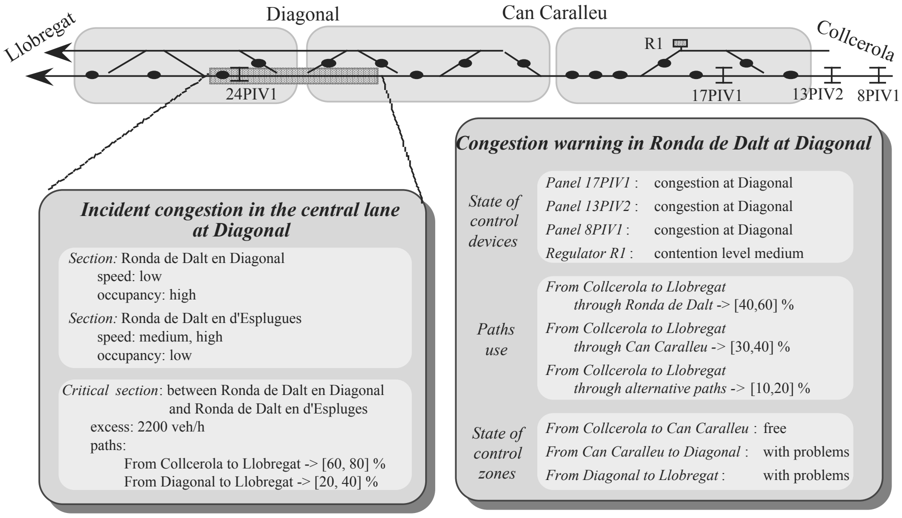


Fig. 5. Possibility functions to abstract traffic values



Incident congestion in the central lane at Diagonal

Section: Ronda de Dalt en Diagonal
speed: low
occupancy: high

Section: Ronda de Dalt en d'Esplugues
speed: medium, high
occupancy: low

Critical section: between Ronda de Dalt en Diagonal and Ronda de Dalt en d'Espluges
excess: 2200 veh/h
paths:
From Collcerola to Llobregat -> [60, 80] %
From Diagonal to Llobregat -> [20, 40] %

Congestion warning in Ronda de Dalt at Diagonal

State of control devices	*Panel 17PIV1* :	congestion at Diagonal
	Panel 13PIV2 :	congestion at Diagonal
	Panel 8PIV1 :	congestion at Diagonal
	Regulator R1 :	contention level medium
Paths use	*From Collcerola to Llobregat through Ronda de Dalt* -> [40,60] %	
	From Collcerola to Llobregat through Can Caralleu -> [30,40] %	
	From Collcerola to Llobregat through alternative paths -> [10,20] %	
State of control zones	*From Collcerola to Can Caralleu* :	free
	From Can Caralleu to Diagonal :	with problems
	From Diagonal to Llobregat :	with problems

Fig. 6. An example scenario

4 Conclusions

In this paper we have argued in favour of an agent-based approach to the construction of advanced Decision Support Systems. We have introduced the concept of an Intelligent Decision-making Assistant (IDEA), and outlined how agent-oriented knowledge and software engineering techniques can be used to guide the process of a principled construction of IDEAs based on a (multi-)agent architecture. Our approach has been illustrated in the road traffic management domain.

We are aware of the fact that an IDEA-based design of DSS puts much burden on the agentsí knowledge models, which leads to rather ìfatî agents and puts limits to the scalability of the approach in this present shape. This is especially true for our traffic management agents, as each of them is to be endowed with domain knowledge to manage a complete problem area. Still, this seems hard to avoid, as real-world traffic engineers conceive traffic flows in this manner, and the decision support dialogues are to follow their lines of reasoning, so as to provide explanations that are understandable, and acceptable, to human operators. Furthermore, the decentralised coordination mechanism of the TRYSA$_2$ system already allows for a significant reduction in the need for coordination knowledge. Nevertheless, improving the scalability of our approach is a major line of future work.

In addition, we are current working on a tighter integration of the different phases of engineering IDEAs. For this purpose, particular attention is paid to the extension of the knowledge-oriented methods for design that have been outlined in this paper [4]. The principled and structured construction of the domain knowledge bases of IDEAs will be complemented by additional structured models (e.g. deeper interaction models, acquaintance models, strategic models etc.) which will eventually lead to a particular (multi-)agent architecture for IDEAs. We are particularly interested in different methods that support the design of structured interaction plans that allow for a dynamic configuration of the questions and answers produced during a decision support dialog between user and IDEA [18]. Decision support for road traffic control in an urban motorway network in the Basque Country, as well as for the management of bus fleets in Southern Spain, are target domains for the evaluation of this approach.

References

[1] Breuker, J.; van de Velde, W. (1994): *CommonKADS Library for Expertise Modelling*. IOS Press.

[2] Brown D.C., Chandrasekaran B. (1989): *Design Problem Solving*, Morgan Kaufmann.

[3] Chandrasekaran B. (1983): Towards a Taxonomy of Problem Solving Types. *A.I. Magazine 4 (1)*, 9ñ17.

[4] Cuena J., Ossowski S. (1999): Distributed Models for Decision Support. In: Weifl (ed.): *Multi-Agent Systems ó A Modern Approach to DAI.* MIT Press, 459ñ504.

[5] Cuena, J.; Hern· ndez, J.; Molina, M. (1996): Knowledge-Oriented Design of an Application for Real Time Traffic Management. In: *Proc. Europ. Conf. on Artificial Intelligence (ECAI-96),* Wiley & Sons, 217ñ245.

[6] French, S. (2000): *Decision Analysis and Decision Support*. John Wiley & Sons.

[7] Hern· ndez, J.; Ossowski, S.; GarcÌa-Serrano, A. (2002): Multiagent Architectures for Intelligent Traffic Management Systems. *Transportation Research C*, Kluwer.

[8] Hern· ndez, J.; Serrano, J. (2000): Environmental Emergency Management Supported by Knowledge Modelling Techniques. *AI Communications 14 (1).*

[9] Iglesias, C.A.; Garijo Ayestar· n, M. (1999): UER Technique - Conceptualisation for Agent Oriented Development. In: *Proc. Intl. Conf. on IS Analysis and Synthesis (ISAS'99)*.

[10] Iglesias, C.A.; Garijo Ayestar· n, M.; Gonz· lez, J.C. (1999): A Survey of Agent-Oriented Methodologies. In: *Intelligent Agents V*. Springer-Verlag.

[11] Jacobson, I.; Booch, G.; Rumbaugh, J. (1999): *The Unified Software Development Process*. Addison-Wesley.

[12] Kinny, D.; Georgeff, M. Rao (1996): A. A methodology and modelling technique for systems of BDI agents. In: *Agents Breaking Away*, Springer-Verlag.

[13] Kirschfink H. (1999) Collective Traffic Control in Motorways. Tutorial at the *11th EURO-Mini Conference on AI in Transportation Systems and Science*. Helsinki.

[14] Klein, M.; Methlie, L. (1995): *Knowledge-Based Decision Support Systems*. John Wiley.

[15] Malone, T.; Crowston, K. (1994): The Interdisciplinary Study of Coordination. *Computing Surveys 26 (1),* 87ñ119.

[16] McDermott J.(1988): Preliminary Steps Toward a Taxonomy of PSMs. In: Marcus (ed.), *Automating Knowledge Acquisition for Expert Systems*, Kluwer.

[17] Ossowski, S. (1999): Co-ordination in Artificial Agent Societies, Springer-Verlag.

[18] Ossowski, S.; Serrano, J.M. (2001): Agent-based Architectures for Advanced Decision Support. In: *Proc. Workshop on Intelligent Physical Agents (WAF)*, URJC.

[19] Silver, M. (1991): *Systems that Support Decision Makers*. John Wiley & Sons.

[20] Vlahavas, I. et al (2002): An Intelligent Multiagent System for WAN Management. *IEEE Intelligence Systems 17(1),* 62ñ72.

[21] Wooldridge, M.; Jennings, N.; Kinny, D. (2000): The Gaia Methodology for Agent-oriented Analysis and Design. *Autonomous Agents and Multiagent Systems 3(3)*. Kluwer Academic Publishers, 285ñ312.

Co-ordinating Heterogeneous Interactions in Systems Composed of Active Human and Agent Societies

Konstantinos Prouskas and Jeremy Pitt

Intelligent and Interactive Systems Group
Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine
Exhibition Road, London SW7 2BT, U.K.
{k.prouskas,j.pitt}@ic.ac.uk
tel. +44 (0)20 7594 6318

Abstract. This paper describes the specification and implementation of the middle layer in a new three-layer time-aware agent architecture. This architecture is designed for applications and environments where societies of humans and agents play equally active roles, but interact and operate in completely different time frames. The middle layer, called the Time-Aware Layer, uses services of the underlying real-time layer to co-ordinate the heterogeneous interactions present in composite human-agent systems. Interactions are unified by abstracting away from their temporal representation, temporal scale and class of parties they involve (be they humans or agents). To achieve this, this paper firstly introduces Availability Functions as the primary mechanism of reasoning about temporal constraints placed on interactions. It subsequently describes their stylised analytic representation and develops a Selective Sampling Algorithm which allows searching through them in bounded time. The resultant implementation allows more effective engineering of the topmost application layer firstly by providing an abstract, unified view of interactions and secondly by predicting and guaranteeing their initiation and completion times.

1 Introduction

Many agent systems operating in the real world affect or depend on humans for their correct and effective operation. In such systems, agents have temporally constrained goals and need to interact with other *agents* as well as *humans* in complex patterns in order to achieve them. Their temporal behaviour is determined by the set of temporal constraints placed equally on the computations and the interactions within the system.

In this paper we describe the specification and implementation of the middle layer in a new three-layer *time-aware* agent architecture. Time-aware agents are defined as agents capable of dealing with the hard, fast temporal dimension of agent-to-agent interactions while equally handling the much softer and slower

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 199–212, 2003.

interactions with humans. This architecture is designed for applications and environments where societies of humans and agents play equally active roles, but interact and operate in completely different time frames.

We focus on the architecture's Time-Aware Layer, which uses services of the underlying real-time layer to co-ordinate the heterogeneous interactions present in composite human-agent systems. To facilitate this co-ordination, interactions are unified by abstracting away from their temporal representation, temporal scale and class of parties they involve (be they humans or agents).

To achieve this, we firstly introduce Availability Functions as the primary mechanism of reasoning about temporal constraints placed on interactions. We subsequently describe their stylised analytic representation within the layer and develop a Selective Sampling Algorithm which allows searching through them in bounded time.

Section 2 provides background information and describes our motivation for this work. Section 3 presents an overview of the overall time-aware architecture and Section 5 focuses on its Time-Aware Layer in which interaction co-ordination takes place. Section 6 performs a brief evaluation before conclusions are presented in Section 7.

2 Background & Motivation

2.1 Composite Human-Agent Systems

Practical use of agent systems has seen their application in areas where there is extensive interaction with humans at the most basic level and societal or organisational structures more generally. Interactions between agents and humans play a key role in areas such as agent-enhanced workflow and e-commerce.

Such systems comprise two classes of entities: agents and humans, each organised in their respective society (Figure 1). Normally, the agent society is formed by a software multi-agent system, while the human one reflects the roles and relationships of the 'real' human societal structure.

Each society is not passive and does not function independently from the other, but forms an active and integral element of the environment. As an example of this, consider the following envisaged agent scenario [1]:

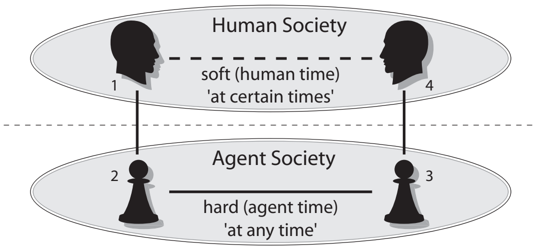


Fig. 1. Composite systems consisting of agent and human societies

The scenario takes place at a doctor's office in which a handheld agent is asked to produce a schedule for a physiotherapy treatment. The agent engages in conversation with the doctor's agent to retrieve the prescribed treatment and subsequently contacts a series of provider agents to determine the best one. After a few minutes the agent returns a proposed schedule of treatment to its human owner. Unfortunately, the schedule is unsatisfactory because it involves the owner driving across town during rush hour. Another agent is then given a stricter set of constraints and asked to come up with an alternative schedule. The second agent returns a better-suited schedule, albeit one that requires its human owner to cancel some of their less important appointments.

The system is initially unsuccessful in its attempt to provide a solution because it has not taken into account that the affected party is a human and therefore bound by a wholly different set of temporal constraints (in this case, the concept of rush hour) than a software agent.

Furthermore, there is unnecessary iteration because humans are never treated as active parts of the system. Instead, they are external, passive elements: the system is a black box in which humans feed some input and receive some output. This results in a trial-and-error approach: problems are not detected until the results are produced, at which point a human has to make modifications and retry.

To avoid these limitations it is therefore desirable to treat both humans and agents as active parts of these systems.

2.2 Co-ordinating Heterogeneous Interactions

The challenge of making agents and humans equally active participants lies in the fundamentally different temporal nature of interactions that it would encompass: agent-agent interactions (within the agent society), human-human interactions (within the human society), and also human-agent interactions in both directions (originating at either society).

From a temporal viewpoint, agent-agent interactions are well defined, fast and predictable, while human-agent interactions are very loosely defined and constrained and operate on an entirely different temporal scale. Human-human interactions are even more loosely defined as they introduce human uncertainty at both ends.

Agent-agent interactions normally assume availability of both parties (they can occur 'at any time') and response times in the order of seconds at most. Additionally, they are commonly iterative in nature and follow to a greater or lesser degree some protocol convention. Human-agent interactions are constrained either by physical location limitations, as when a human is 'out of reach' of an agent, or by higher-level concepts such as attention, interest, mood etc. (that is, they only occur 'at certain times'). Further factors that may affect human-agent interactions include day/night-time, time of day, working hours, bank holidays, illness, social commitments, biological reasons etc. Human-human interactions are a generalisation of human-agent interactions where both parties are human, and are the least predictable of the three.

This discrepancy stems partially from the difference in which agents and humans understand and deal with time. Agents, being principally software computing entities, have an implementation-level representation of time such as vector clocks commonly found in distributed systems, whereas humans are used to dealing with abstract relative concepts such as 'today' and 'tomorrow'. Agents also have a much finer-grain representation of time like 'seconds elapsed since January 1st, 1970', whereas humans normally only need a much coarser granularity like '3 o'clock' or 'this afternoon'. Finally, agents have well-defined means of measuring temporal intervals (by measuring the difference of two UTC-synchronised clocks, for example), whereas humans rely on a variety of calendar systems and intervals of variable duration such as 'evening', 'month', 'year' and so on.

2.3 Related Work

There are several examples of *real-time agent architectures*, both 'hard' [2, 3] and 'soft' [4]. Real-time agent systems are systems in which agents are temporally constrained in achieving their goals. They combine real-time (RT) and artificial intelligence (AI) characteristics by either embedding AI in RT, embedding RT in AI, or by adopting a co-operative RT-AI approach [5].

However, applying these architectures to this particular problem presents two key limitations. The first is that they mainly consider systems comprised of a single class of entities, namely software agents. Adding a human class, however, is problematic as humans 'are not real-time', that is, they cannot be considered to be the well-defined, predictable, principally computational entities their agent counterparts are.

The second limitation is that these architectures assume the internal deliberative process of an agent is exposed and focus on temporally constraining it in real time. In extending this to the type of composite human-agent systems we are considering, however, expressing temporal constraints on the deliberative process of a human actor is not appropriate. This is similar to deontic specifications in software engineering, where placing obligations on human actors is equally inappropriate.

On the other hand, work on *collaborative systems* between agents and humans either does not address temporal constraints [6] or, when it does [7], it is not from a real-time, predictability perspective. There are also examples of "dramatic failures" in such systems [8] because the temporal distinction between humans and agents has not been made explicit in the design.

There is therefore a need to identify the different classes of parties present; secondly to reason about how their interactions (rather than their deliberative processes) affect the system's temporal behaviour; and finally be able to handle all interactions in a uniform manner, irrespective of the parties they involve.

3 A Time-Aware Agent Architecture

We have developed a real-time architecture, comprising *time-aware* agents, that better meets the requirements described above. Time-aware agents are defined as agents capable of dealing with the hard, fast temporal dimension of agent-to-agent interactions while equally handling the much softer and slower interactions with humans. Time-aware agent systems can deal with an amalgam of hard, soft, human and non-real-time interactions, reason about the temporal constraints placed on the system by each type of interaction, make transformations between them and co-ordinate (schedule) activities seamlessly irrespective of their constituent constraints.

In this architecture, we consider systems comprised of a number of *parties*, agents or humans, logically or physically distributed over an interconnected network environment such as the Internet. Agents are lightweight computational entities that cannot share any data except by means of interaction through message passing. They are trustworthy and work co-operatively toward achieving the system's goals.

We primarily consider the co-ordination of agent-agent and human-agent interactions. Human-human interactions are considered only as far as they are agent-mediated, that is, emergent from a specific pattern of interaction (namely $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ in Figure 1).

We treat deliberation as internal to both agents and humans, treating both opaquely. We instead focus on providing the environmental and architectural mechanisms by which agents can be aware of the different types of interactions they are engaged in, as well as their temporal repercussions in achieving their goals.

We have realised this architecture using a prototype implementation in the Agent Process Interaction Language (April) [9], consisting of three layers: the *April Real-Time Run-Time* layer, the *Time-Aware Layer* (TAL) and the *Application Agent Layer* (AAL). The ART layer forms the underlying real-time agent platform. Similar to a real-time operating system, the ART provides temporal guarantees to the rest of the architecture. More information on the ART can be found in [10]. The AAL is the overlying application layer; in it, software agents exist and work toward their goals. The discussion in this paper focuses on the middle of the three layers: the Time-Aware Layer.

4 The Time-Aware Layer

The Time-Aware Layer (TAL) is the layer that provides a unified temporal view of the entire system. The TAL abstracts away from different time representations, disparate temporal scales and class of parties (human or agent) engaged in interactions. The main components of the TAL are (Figure 2):

Human/Agent Handles: Both humans and agents are uniquely identified by their *handle*. Handles are an abstraction mechanism by which parties in the system can be uniformly addressed.

Task Representation: Tasks are hierarchically decomposed sequences of either *processing* or *interacting* (communicating).

Unified Time Representation: The Unified Time Representation (UTR) is an April macro layer responsible for transforming different human and agent

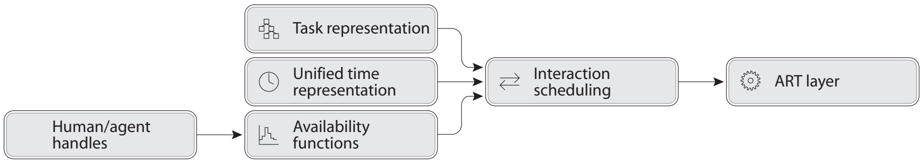


Fig. 2. The Time-Aware Layer

time representations, units, and calendars into a common representation (namely, Universal Coordinated Time). Human concepts like time zones, working hours, next working day, etc. are handled by the UTR.

Availability Functions: Availability Functions are associated with each party to express that party's degree of presence within the system. They are fully described in Section 5.1.

Interaction Scheduling: Interaction scheduling involves co-ordinating and guaranteeing initiation and completion times of interaction sequences (generated from tasks). Interaction scheduling is detailed in Section 5.2. Schedules generated by this process are passed on to the underlying ART layer.

5 Realisation

Interaction scheduling is central to the TAL and Availability Functions play a key role in this process. In order to make interaction scheduling fast and predictable enough for real-time application, we constrain the Availability Function representation and discuss a Selective Sampling Algorithm.

5.1 Availability Functions

The TAL uses the concept of *availability* and *Availability Functions* (AFs) to reason about the temporal nature of interactions. The availability of a party i at time t, given by $V_i(t)$, is that party's degree of 'presence' in the system, as far as interactions with other parties are concerned.

For agents, availability is determined by the intervals during which they are reachable via the network. Some network connections (such as local area network links) may be persistent, others (such as satellite links) may be intermittently connected for fixed periods of time, while yet others (such as dial-up links) may have connection times and intervals which can only be statistically characterised.

For humans, availability is principally affected by the time spent sitting in front of their terminal (for fixed contact points) or the times they are in the vicinity of the relevant device (for mobile contact points such as cellular phones). Human AFs closely follow the natural daily cycle.

AFs are mathematically expressed as probabilities. However, AFs are not probability distributions. They rather function as look-up tables that provide a statistical measure of availability at any point in time.

AFs are assumed to be externally specified by either the relevant parties themselves or a third party such as the application programmer. Realistic AF generation is important as it directly affects the quality of the predictions, but is outside the scope of this research. Nevertheless, we have used screensaver data to generate experimental human AFs for validation purposes. Simplified versions of this data have been used to produce the figures presented in this paper.

5.2 Interaction Scheduling

Interaction scheduling is the process of co-ordinating the system's interactions such that the time of their completion is known or can be statistically predicted and guaranteed in advance. Interaction scheduling (in the TAL) is the counterpart of computation time scheduling (in the ART).

A one-way interaction (a notification) *completes* when that notification is received *and* the recipient becomes aware of it. Similarly, a two-way interaction (a request-reply) completes when the reply reaches the initiator *and* the initiator becomes aware of it. We therefore distinguish between the time a message is delivered from the time it is handled.

AFs drive not only the completion time of an interaction, but also the time of its initiation. For a guarantee level g (expressed as a probability) and a party i, an interaction can be initiated *or* completed only when $V_i\left(t\right) \geq g$.

For scheduling interactions interspersed with processing, we also distinguish between *off-line* and *on-line* processing. Off-line processing at a party can proceed independently of its availability, while on-line processing can proceed only when availability allows it, that is, when $V_i\left(t\right) \geq g$. For example, a request for a human to review a paper can proceed off-line, while requests for e-mail composition are normally done on-line.

Figure 3 shows an example of interaction scheduling for a typical request-reply scenario for $g = 0.5$. At $t = t_1$, some off-line processing has completed at j's end and a request is ready to be sent to k. This interaction will not be initiated until $t = t_2$, when $V_j\left(t_2\right) \geq 0.5$. Party k is not aware of this until $t = t_3$, when $V_k\left(t_3\right) \geq 0.5$. As a result of the request, some on-line processing needs to be performed, proceeding only when $V_k\left(t\right) \geq 0.5$. At $t = t_4$ processing is complete. The reply can be initiated immediately, however it will not be handled by j until $t = t_5$, when $V_j\left(t_5\right) \geq 0.5$. Interaction scheduling can be conceptually reduced to repeated applications of variations on a primitive operation mathematically formulated as:

> *Given a function $f\left(t\right)$, a reference point $t = t_{ref}$ and a condition C on $f\left(t\right)$, find:*

$$t_x : \left(t_x \geq t_{ref} \wedge C\left(f\left(t_x\right)\right)\right) \wedge \nexists t_x^{'} : \left(t_x > t_x^{'} \geq t_{ref} \wedge C\left(f\left(t_x^{'}\right)\right)\right) \qquad (1)$$

Practically, arbitrarily-shaped functions $f\left(t\right)$ can either be stored *symbolically* in their analytic form, or *numerically* as discrete data samples. In our context, sampling has two undesirable properties:

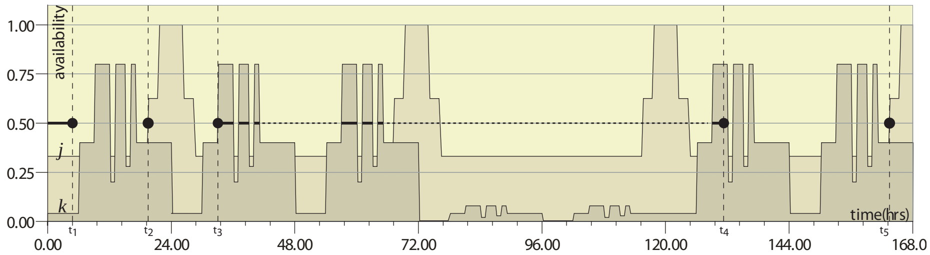


Fig. 3. Interaction scheduling for a request-reply scenario between parties j and k, and a guarantee level $g = 0.5$

- The success of sampling is dependent on a good choice of sampling frequency (step size). A large step may result in solutions landing between sampled points. A small step causes unnecessary over-sampling leading to unfeasibly large computation times. Unfortunately, in composite human-agent systems, we are dealing with interactions with greatly disparate temporal scales. On one hand, it is desirable to detect sub-second availability periods for agents, while on the other, it may be several hours or even days before human availability rises to the desired level. There is therefore no single best step size.
- Sampling is an iterative process which cannot be temporally bound without restricting the search range. This has adverse implications for any real-time system attempting to provide deterministic guarantees. A match may be found immediately or it may be, for example, that a human is on vacation and several weeks must be searched through until one is found.

To address this we have developed a stylised analytic representation for AFs (Section 5.3) which, when used in conjunction with a Selective Sampling Algorithm (Section 5.4), allows searches to be performed in bounded time.

5.3 Availability Function Representation

AF representation forms the first half of making interaction scheduling fast and predictable enough for real-time application. We constrain AF representation in the following ways:

- AFs are analytically stored and represented as hierarchical operator trees named *environments*. Primitive (leaf) nodes of environments are named *spans*. For example, a ‘working’ span for a human might set the availability to 0.9 whereas a ‘sleeping’ span might set the availability to zero.
- Spans have a *constant* availability value v.
- The periods of composed environments must be harmonics (integral multiples of one another). This is a requirement for the Selective Sampling Algorithm to work correctly. Non-harmonic environments must be broken down

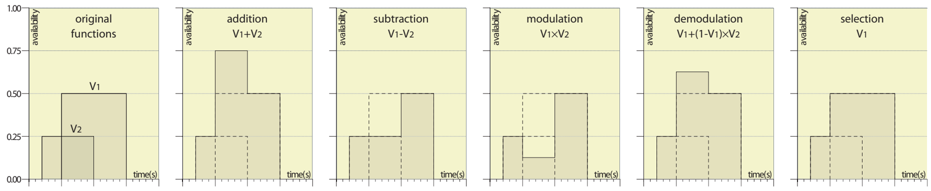


Fig. 4. Results of Availability Function composition for common composition functions f (addition, subtraction, modulation, demodulation, selection)

into a series of sub-environments such that they have harmonic repetitions. Fortunately, most human environments follow the natural daily cycle and therefore do not require any transformation.

Environments go through a series of operations (dictated by the nodes in the tree) to arrive at the final composite AF. Nodes in the AF trees can be one of the following types:

Span (v): Sets the availability value to $v \in \mathbb{R}$.[1]
Bound (V, i): Bounds the value of environment V to the interval $i = [t_1, t_2)$. V remains unchanged within $[t_1, t_2)$ but becomes undefined outside it. Undefined environments do not affect the availability value.
Repeat (V, p): Repeats environment V with a period of p seconds.
Offset (V, o): Translates (offsets) environment V by o seconds to the right.
Compose $(f, V_1, V_2, \ldots)$: Composes two or more environments together using the function f as an infix operator. Composition only has an effect at points where at least two environments are defined. Function f can be any binary function, but it is normally one of addition $(= V_1 + V_2)$, subtraction $(= V_1 - V_2)$, modulation $(= V_1 \cdot V_2)$, demodulation $(= V_1 + (1 - V_1) \cdot V_2)$ or selection $(= V_1)$.

Environment composition for common functions f is illustrated in Figure 4. Repeated compositions result in more complex AFs, such as the ones shown in Figure 5 (the notation in this figure has been simplified for clarity). Given the AF's tree representation, an evaluation function is provided to return $V(t)$ at any point t. Simplified pseudocode for evaluating the AF `node` at point `t` is given in Algorithm 1.

5.4 Selective Sampling Algorithm

The task of the Selective Sampling Algorithm (SSA) is to generate a minimal set of sampled points while guaranteeing that, if the solution of (1) exists, it will be among those points.

[1] Note that the value of v is not constrained at this stage. This allows for a simpler tree representation of AFs. However, since AFs represent probabilities, final results are clipped to lie within $[0, 1]$.

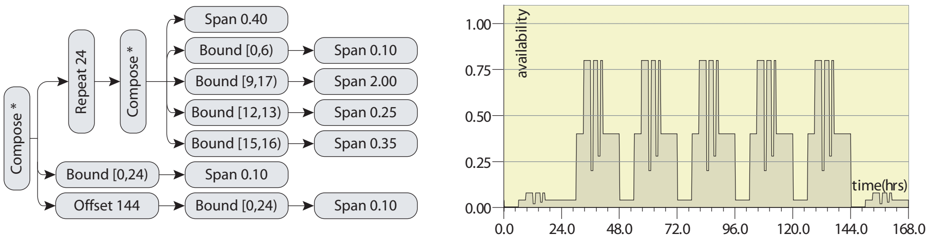


Fig. 5. Tree representation and resultant Availability Function

Informally, the SSA operates on the premise that, since spans are constant, only their (bounded) extremities are relevant. Furthermore, repeated environments generate relevant points for only their first full period – any solution found in subsequent periods will have a corresponding solution in an earlier period. Lastly, environment compositions are periodic with a period that is the maximum of the periods of their constituent environments.

The SSA can be specified as follows:

Input: An AF $V_i\,(t)$ and a reference point $t = t_{ref}$ at which the search starts.

Output: A set of time points where spans may cause a change in the availability value. It can be shown that this is a superset of the set of points $\mathcal{S}$ that satisfy (1) for each discrete value that $V_i\,(t)$ can take, and that if a solution t_x exists, then it is always contained in $\mathcal{S}$.

Steps: There are two stages. In the first stage, a set of *relevant points* is generated from t_{ref}. A point is relevant if it is the earliest point where a change in spans may cause a change in the availability value. The reference point is always relevant. The theorems which drive relevant point generation for a reference point t_{ref} are summarised below (proofs are omitted):

1. Span (v) generates a single relevant point at $t = t_{ref}$.
2. Bound (V, I) generates a relevant point at each edge of I.
3. Repeat (V, p) generates relevant points only in its *relevant* period. The relevant period is the first full period nearest to t_{ref} in the direction of the search.

Algorithm 1 Availability Function evaluation algorithm

```
evaluate(node, t)
{
  case node of type
    Span(v):  v
    Bound(V,I): evaluate(V,t) if t lies in I, undefined otherwise
    Repeat(V,p): evaluate (V,remainder(t/p))
    Offset(V,o): evaluate (V,t-o)
    Compose(f,V1,V2,...): evaluate(V1,t) f evaluate(V2,t) ...
}
```

4. Offset (V, o) has a relevant point at t_x if and only if the environment V has a relevant point at $t_x - o$.
5. Compose $(f, V_1, V_2, \ldots)$ generates relevant points which always lie in the relevant periods of its constituent environments $(V_1, V_2, \ldots)$.

In the second stage, each of the relevant points of the first stage functions as a reference point to add its own relevant points to the set. This stage accounts for possible availability values resulting from the composition of two or more spans.

After the second stage, the list of relevant points can be passed to the evaluation function to test (sample) each point. Figure 6 illustrates relevant point generation in the first and second stages of the SSA. Based on this specification, correctness, completeness and complexity properties of the SSA can be proven. We omit the full format proof of these results here, however we present a sketch of the computational complexity result as it is the limiting factor for time-aware agents.

The first stage is $\mathrm{O}(n)$ with the number of nodes in the AF representation, generating a linearly proportional number of relevant points (a maximum of two per node). The second stage is identical to the first, only t_{ref} takes values from the results of the first stage. The total algorithmic complexity is therefore $\mathrm{O}(n^2)$. Theoretically, up to $4n^2$ points will need to be sampled in the worst case, although practically not every node will generate relevant points and many points generated from the first and second stages will overlap. The main advantage of the SSA is therefore that it completes in bounded time.

Another advantage is that, as it operates on analytic representations of AFs, it will find a solution (if one exists) irrespective how slim or far into the future it is. Agent functions that change in a temporal scale of milliseconds can be handled in the same way as human functions that may be specified in terms of hours.

Searching in 'forward' time can be used to predict the completion time of current interactions, while searching in 'backward' time can be used to determine the latest start time for an interaction sequence such that it completes by a specified deadline.

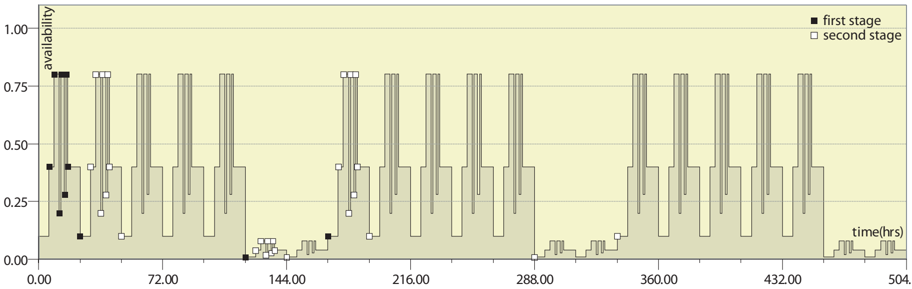


Fig. 6. Relevant point generation in the first and second stages of the Selective Sampling Algorithm for a reference point at $t = 0$

The SSA can be used as the basis for a wide range of interaction scheduling operations, including *(i)* finding the first point at which availability exceeds or falls below a threshold *(ii)* finding the maximum and minimum value of availability within an interval *(iii)* summing the length of time for which the availability lies above or below a threshold within an interval and *(iv)* calculating the value of functions of the form $h = \int_{t_1}^{t_2} (V_i(t) - g)\,\mathrm{d}t$, which give a measure of how much 'better' or 'worse' availability is during $[t_1, t_2]$ with respect to g.

6 Evaluation

The SSA and the encompassing TAL layer will be evaluated from two distinct perspectives: raw real-time performance will be addressed first, followed by a brief discussion of experimental results in a practical application.

There are two versions of the SSA: one implemented in April and one in C (still accessible through April). The April version executes much slower than its C counterpart. However, its main advantage is that it is interruptible and resumable at any point by the April virtual machine. Other advantages are that Compose nodes are much more flexible, allowing any composition function (standard or user-defined) and the composition of an arbitrary number of environments in the same node.

The C version is faster than the April one by approximately 350 times. This improves the minimum temporal scale that can be handled and allows much more complex AFs without adversely affecting the system's performance. However, once the C algorithm is started, it must execute to completion. Another limitation is that Compose nodes are restricted to pre-defined composition functions and only two environments can be composed in a single node.

Experimentally, the number of relevant points generated on average by the SSA was much lower than the worst case, lying between 1% and 2% of the theoretical $4n^2$ limit. In absolute terms, relevant point generation is very fast, with the C version generating relevant points for a 50-node AF within 160μs on a 1GHz processor. Evaluating the same function at each point takes approximately 1.2μs per point.

We are also in the process of applying this architecture in a workflow management system to schedule typical processes in Institutions of Higher Education, where some interactions are scheduled with millisecond accuracy, while the overall temporal scale can be of the order of years.

Figure 7 shows interaction scheduling for filling in a six-month progress report. Participating parties are shown on the vertical axis. Solid bars denote that a local party process is executing, dashed bars denote that the process is blocked waiting for its dependencies to be satisfied. Arrows indicate messages (interactions) between parties. Left-pointing triangles indicate the delay in interaction completion (as determined by availability), while right-pointing triangles indicate the delay in its initiation. Availability functions are plotted immediately below their corresponding party. It can be seen how certain interactions occurring over weekends will not be initiated or completed until the next working day

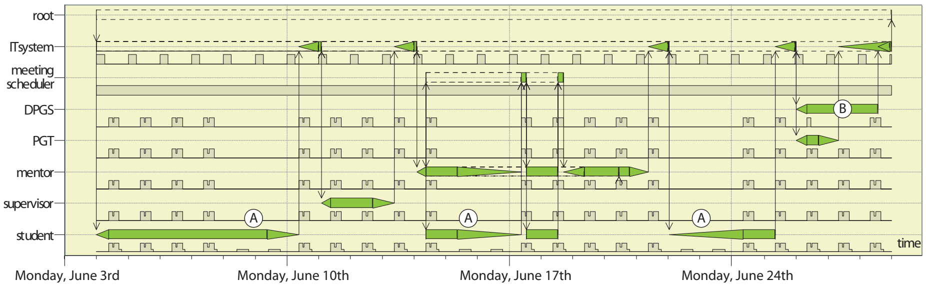


Fig. 7. Interaction schedule for a “progress report” workflow process, showing (A) interactions scheduled around weekends and (B) processing time affected by periods of unavailability

(labelled as A in the figure), as well as how on-line processing time is affected by periods of unavailability (labelled as B). This application has also demonstrated that prediction quality is directly affected by how accurately AFs reflect ‘real’ availability.

The TAL and the encompassing time-aware architecture represent ongoing research work and the current iteration has several limitations:

- The architecture is unsuited to hard real-time environments. Guarantees associated with schedules generated by the TAL are very soft and should be treated as guidelines only. This is a consequence of the presence of interactions which wholly or partly depend on humans for completion.
- In all calculations, message delivery latency is assumed to be negligible. This is a valid assumption for local controlled network environments but is not necessarily true for large-scale distributed systems over the Internet.
- Commitments to tasks may translate to changes in the AFs. For example, availability may be decreased for the duration of a task so that other tasks' interactions are scheduled around it. These changes are currently done on a FIFO basis. This may cause a lower-priority task to be scheduled in preference to a higher-priority one. The selective re-scheduling of a minimal set of affected tasks to avoid these priority inversions has not yet been addressed.

7 Conclusions

We have presented the specification and realisation of the Time-Aware Layer in a time-aware architecture which co-ordinates agent-to-agent and human-to-agent interactions. To facilitate this co-ordination, interactions are unified by abstracting away from their temporal representation, temporal scale and class of parties they involve. This is achieved by a combination of a Unified Time Representation, common human/agent identifiers and Availability Functions.

We have used Availability Functions to model and predict the initiation and completion time of any kind of interaction. By restricting the Availability Function form with a stylised analytic representation and by taking advantage of their piecewise constant and repetitive nature, we have developed a Selective Sampling Algorithm which allows searching through them in bounded time.

The research work described in this paper is ongoing. There are still several outstanding issues, such as the effects of wide-scale distribution and the interplay between computational and interactional scheduling, which will need to be addressed in the near future. Nevertheless, initial evaluation results have been encouraging and we are applying the described time-aware architecture to implement a workflow management system.

Acknowledgements

This work has been undertaken with the financial support of the EPSRC-funded project TARA^2 (GR/N24940).

References

[1] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001) 35–43 200

[2] Musliner, D., Durfee, E., Shin, K.: CIRCA: A Co-operative Intelligent Real-time Control Architecture. IEEE Transactions on Systems, Man, and Cybernetics **23** (1993) 1561–1574 202

[3] Botti, V., Carrascosa, C., Julian, V., Soler, J.: The ARTIS Agent Architecture: Modelling Agents in Hard Real-Time Environments. In: Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Springer-Verlag (1999) 63–76 202

[4] Wagner, T., Garvey, A., Lesser, V.: Criteria-Directed Heuristic Task Scheduling. International Journal of Approximate Reasoning, Special Issue on Scheduling **19** (1998) 91–118 202

[5] Musliner, D., Hendler, J., Agrawala, A., Durfee, E., Strosnider, J., Paul, C.: The Challenge of Real-Time Artificial Intelligence. Computer **28** (1995) 58–66 202

[6] Babaian, T., Grosz, B., Shieber, S.: A Writer's Collaborative Assistant. In: Proceedings of the International Conference on Intelligent User Interfaces. (2002) 7–14 202

[7] Payne, T., Lenox, T. L., Hahn, S. K., Sycara, K., Lewis, M.: Agent-Based Support for Human/Agent Teams. In: CHI 2000 Conference on Human Factors in Computing Systems, ACM Press (2000) 202

[8] Chalupsky, H., Gil, Y., Knoblock, C. A., Lerman, K., Oh, J., Pynadath, D. V., Russ, T. A., Tambe, M.: Electric Elves: Agent Technology for Supporting Human Organizations. Artificial Intelligence **23** (2002) 11–24 202

[9] McCabe, F., Clark, K.: April: Agent Process Interaction Language. In Jennings, N., Wooldridge, M., eds.: Intelligent Agents. Volume 890. Springer-Verlag (1995) 203

[10] Prouskas, K., Pitt, J.: Towards a Real-Time Architecture for Time-Aware Agents. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems. Volume 1. (2002) 92–93 203

SABPO: A Standards Based and Pattern Oriented Multi-agent Development Methodology

Oğuz Dikenelli and Riza Cenk Erdur

Ege University, Department of Computer Engineering
35100 Bornova-Izmir, Turkey
{oguzd,erdur}@staff.ege.edu.tr

Abstract. Organizational metaphor, in which each agent plays a specific role to achieve organizationís global goal(s), seems as the most suitable approach to model multi-agent systems. In this paper, a new methodology, which uses organizational metaphor as its basis and integrates this metaphor to the FIPA standards and well-known interaction protocols in a systematic way, is introduced. In addition to the proposed methodology, a new interaction pattern, which is called as ìontology inceptionî, is introduced. This pattern illustrates how the methodologists can mine new agent interaction patterns from the requirements and how these patterns can be inserted into the FIPA standards to create a pattern catalogue. Naturally, interactions of each interaction pattern add new tasks to the participant agents. We believe that the requirements of these new tasks and how they are inserted into the agentís internal architecture must be documented as part of the pattern knowledge. Thus, internal requirements of the ìontology inceptionî pattern are discussed from the perspective of the participating agents. By this way, understandability of the pattern will be improved.

1 Introduction

Characteristics of software systems are going to change dramatically. Today, software developers have to design systems to execute in a world where autonomous software components are already executing and locally interacting with each other and with the dynamically changing environment [21]. In such an environment, component abstraction is not sufficient and developers need an abstraction with capabilities such as goal delegation, semantics based interaction, and reasoning to generate knowledge from events and messages [4]. These capabilities are provided by an agent abstraction and systems executing on such environments have to be developed as multi-agent systems. In the last few years, several multi-agent system development methodologies, which take agents as high-level abstractions and develop the methodology around the agent concept, have been proposed. Most of these proposals either extend object oriented (OO) methodologies using agent abstraction instead of

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 213-226, 2003.

object or focus on the internal design of the individual agent architecture [13],[14],[15],[18]. However, MASs behave like social organizations where each agent plays a specific role within the organization to satisfy the organizationís global goals. Hence, we need a different approach that takes organizational abstraction as a key concept to model the multi-agent systems working on dynamic environments.

Gaia [19] is one of the well-known methodologies that uses the organizational view to construct the development methodology. In Gaia, an agent organization is defined by collection of roles, each roleís responsibilities, and interactions between those roles. All of these entities try to satisfy the organization's global goals all together. Hence, Gaia assists the developers to systematically discover the abstract entities of the organization (such as roles, responsibilities, interaction protocols and activities) and to map these entities to the concrete ones (such as agents, services and acquaintances). But, Gaia has been initially developed for modeling only the closed systems, where organizational structure of the system is static and do not change at run-time. It is a very critical shortcoming, since MAS developers are mainly interested in open and dynamic environments.

This shortcoming of the Gaia methodology has been addressed by Zambonelli, Jennings, Wooldridge [22], Omicini [16], and Bernon et. al. [5]. Zambonelli, Jennings and Wooldridge introduced three additional organizational concepts over the Gaia: organizational rules, organizational structures and organizational patterns. Organizational rules are used to define global requirements of the organization. Hence, new agents can learn organizationís global rules when they join the organization in an open environment. Organizational structure defines the architecture of the organization, which is selected among many possible alternatives according to the specified requirements. Organization patterns express pre-defined organizational structures that can be re-used from system to system.

Omiciniís methodology, which is called SODA, explicitly takes the agent environment into account. Environment is modeled as services and agents use these services to work properly within the open environment according to the systemís requirements. SODA also separates the roleís tasks into individual and social ones. Social tasks are used to model the organizational aspects of a MAS.

ADELFE [5] is another methodology that directly focuses on the adaptive nature of the environment. It introduces the concept of ìNon Cooperative Situationsî (NCS) that defines the behaviour of local agents when they encounter an unpredictable situation in a dynamic environment. During the development process, they try to identify such NCS to model local agentsí behaviour.

It is certain that any methodology, which aims to model MASs must have a method to define organizational rules and must explicitly take the environment, organizational structures, and NCS into account. Although the methodologies discussed above propose useful modeling constructs for dynamically changing environments, they do not take standardization efforts of the agent community into account. We believe that standardization is the most important criteria for multi-agent systems executing on dynamic and open environments. The Foundation for Intelligent Physical Agents (FIPA) was established to develop standards for supporting interoperability among agents and agent-based applications. FIPA standards define a run-time environment, an abstract architecture as an organizational structure, an agent communication language, and protocols for semantics based interactions to construct MASs working in open environments. Thus, any attempt of methodology development must take the FIPA standards as the basis, since it already provides the necessary framework for multi-agent development in open and dynamic environments.

Standardization is an evolving process and new requirements of multi-agent systems must be integrated into the FIPA standards. Hence, as an agent community we need a methodology based on FIPA standards and need a method for integrating the new requirements identified during the development process into the FIPA standards. In this study, a new methodology is proposed based on FIPA standards to illustrate our vision. As part of the methodology, we define a interaction pattern identification mechanism which extends FIPA standards by defining new interaction protocols. To illustrate the mechanism of FIPA-compliant interaction pattern identification, we introduce a new pattern called as ìontology inceptionî. This pattern can be used to construct adaptable multi-agent organizations working in the environment, where ontologies change at run-time and new ontologies may be added to the system at any time.

The rest of the paper is organized as follows: Section 2 gives an overview of the SABPO methodology, especially discussing how the FIPA standards are used during the modeling process. The use of the methodology is illustrated by means of a case study in section 3. The ìontology inceptionî pattern is introduced in section 4. This introduction includes the motivation behind the ìontology inceptionî pattern, its interaction mechanism and its affect to the participating agentís internal architecture. Last section concludes the study and outlines main contributions over the previous studies.

2 Overview of the SABPO Methodology

Our approach puts the FIPAís abstract architecture specification [9] to the center of the methodology as a basic organizational structure and tries to tries to create a concrete FIPA architecture that satisfies system requirements. Before discussing the details of the methodology, we need to introduce the elements of the abstract architecture and the relationship between these elements. Fig. 1 outlines the basic relationship between the key elements of the FIPA abstract architecture.

As seen from fig. 1, agents communicate using transport-messages, which are sent or received by message transport service. Each transport-message includes a message, which is written in FIPA agent-communication-language (ACL) and the content of the message is expressed in one of the FIPA content languages. Message-transport-service is not critical from the modeling methodology point of view, since any FIPA-compliant agent framework implements this service as a part of an agentís internal architecture [5],[17]. But defining interactions between the agents is one of the most critical activities of the MAS development.

In FIPA based agent systems, agent interactions are specified using the pre-defined FIPA interaction protocols. Each interaction protocol defines a pre-agreed message exchange between the agents. For example, FIPA Query Interaction Protocol, which defines how one agent may query another agent, is shown in fig. 2. Like all FIPA interaction protocols, this protocol uses FIPA ACL communicative acts (such as query-if, query-ref, inform etc.) within a well defined semantic. Hence, each interaction protocol can be considered as a generic pattern where each participating agent plays a specific role within the defined semantic to solve a particular recurring problem. Since we accept interaction protocols as generic patterns, we use the concepts of the interaction protocol and the interaction pattern interchangeably within the paper.

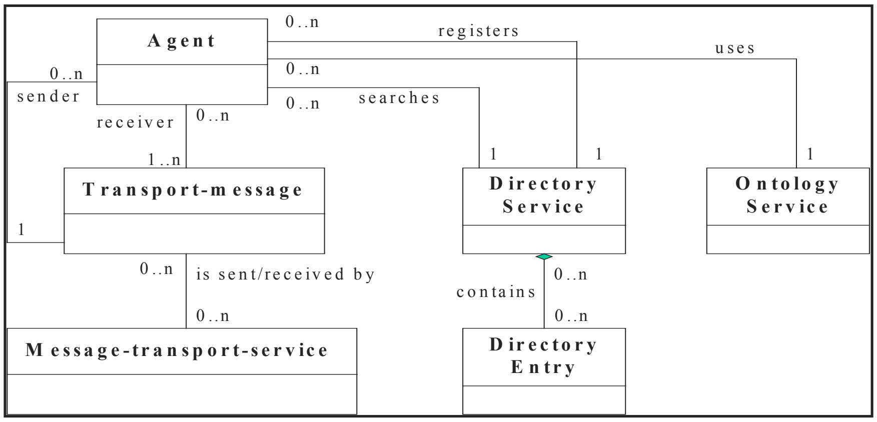


Fig. 1. Basic agent relationships

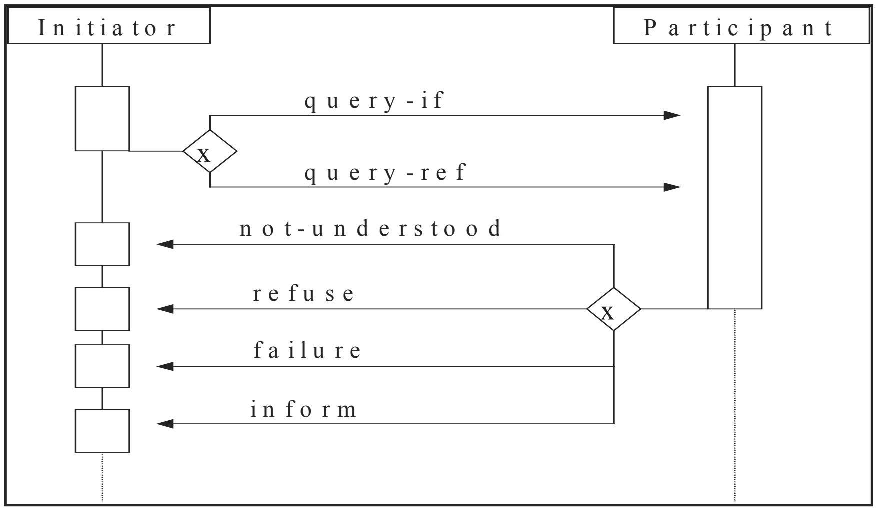


Fig. 2. FIPA-query interaction protocol

Directory service, which is an mandatory component of FIPA abstract architecture, must be analyzed carefully during the modeling process, since this service is the heart of any multi-agent organization running in an open environment. A directory service provides a shared information repository, in which agents may publish their capabilities and in which they may search for other agents of interest. A directory service provides the following standard services:

- An agent may register a directory-entry with the directory service.
- An agent may modify a directory-entry that has been registered with a directory service.
- An agent may delete a directory-entry from a directory service.
- An agent may query a directory service to locate an agent of interest.

In addition to these mandatory elements, FIPA abstract architecture specification defines some optional elements. Ontology is one of these elements. Ontologies provide a vocabulary for representing and communicating knowledge about some topic and a set of relationships and properties that hold for the entities denoted by that vocabulary. Although ontologies are defined as optional elements in the FIPA specification, they are one of the most important issues to be considered during the development of MASs running in open environments. Because, in an open environment, agents can join to or leave from the agent system unpredictably; hence, newly joined agents learn how to communicate with other agent of the organization from the declaratively represented explicit ontologies. Hence, SABPO accepts the ontology service as an mandatory element. That is why ontology service is defined as a part of abstract architecture in Fig1.

Usage of explicit ontologies are defined in another FIPA specification [10]. This specification introduces a new agent called as Ontology Agent (OA). OA provides some ontology service to an agent organization. As well as the other agents, the OA registers its services with the directory service. The OA also registers the list of the maintained ontologies and its translation capabilities, in order to allow other agents to query the directory service for the specific OA that manages a specific ontology. OA provides the following services:

- Helping a FIPA agent in selecting a shared ontology for communication.
- Creating and updating a specific ontology, or only some terms of the ontology.
- Responding to queries for relationships between terms or between ontologies. (Six relationships are defined in the specifications: ontologies can be identified, equivalent, extension of the other, weakly-translatable, stronglyñtranslatable or approximately-translatable. OA stores the defined relationships between the ontologies and agents in the organization can query this knowledge.
- Translating expressions between different ontologies (of course, ontologies must be translatable and translation rules must be defined by an authority.)

Agents query the OA to get the desired service using the FIPA interaction protocols where content of the each ACL within the protocol is written in FIPA-Meta-Ontology.

As a conclusion, SABPO identifies tasks of all abstract elements of the architecture and FIPA based interaction patterns between these elements, to realize a concrete architecture, which satisfies the systemís requirements. SABPO only defines the analysis and design phases of the development process. Models created and activities involved in these phases are described below.

2.1 Analysis Phase

SABPO analysis phase exploit two different models; the role model and the interaction model as in the Gaia methodology. But FIPA abstract architecture and FIPA interaction patterns are taken into consideration during the modeling process.

Role Model. This model mainly defines the roles of the organization and responsibilities of these roles to satisfy the organizationís global goals. These roles can be considered as abstract agent elements of the FIPAís abstract architecture specification. Then, SABPO introduces two new roles based on the abstract architecture specification. These roles are ìDirectory Service Provider î and ìOntology Service Provider î. Services given by the directory service and OA of the abstract architecture are accepted as those roles' responsibilities. In SABPO, role model is documented using the Gaiaís modeling style.

Interaction Model. This model defines the interaction protocols between the agents. To create a FIPA-compliant architecture, ìDirectory Service Provider î and ì Ontology Service Provider î roles participate in interactions using the interaction patterns defined in the FIPA specifications. These interactions are documented using Agent-UML [2] notation and FIPAís communication acts are explicitly indicated on the interaction diagrams. Inclusion of a directory service provider and an ontology service provider forces the developers to make same decisions in the analysis phase. These decisions are:

- Name and general objectives of the global ontology must be identified.
- Relationships between the global ontology and if there are any other known ontologies in the domain, then they must be identified.
- If more than one ontology is identified for the organization, then the specific ontology used for each role must be identified.
- Services given by the directory service provider and the ontology service provider roles during the interactions must be specified according to the related FIPA standards.

2.2 Design Phase

In SABPO design phase, a FIPA based concrete architecture is constructed by transforming the abstract entities identified in the analysis phase to the concrete ones using FIPA specifications. Design phase generates three models which are named as, the ontology model, the agent model and the detailed interaction model.

Ontology Model. This model extends the ontology knowledge derived in the analysis phase. The global ontology can be modeled using any of the well known ontology modeling languages such as DAML [12], extended UML [1] [6], etc. Then, the mappings between this model and FIPA-Meta-Ontology specification are defined to make the global ontology, which is managed by the OA, accessible from outside. If there are any known ontologies in the domain, the mappings between these ontologies and the global ontology are also defined to make OA capable of providing translation service to the organization. At this point, organisational rules are defined using the

Zambonelli, Jennings and Wooldridge's studyís approach and stored as a part of the ontology knowledge.

Agent Model. This model defines the agent types and assigns the roles defined in the analysis phase to the agent types. This model also defines the services of each agent according to the responsibilities of the roles assigned to the agents. Hence, this model combines the "Agent Model" and "Service Model" of the Gaia methodology. ìOntology Service Provideri role and ìDirectory Service Provideri role and their defined services are assigned to both the ontology agent and the directory facilitator agent, which are the agents that are used for these roles in FIPA specifications. Since each agent must adapt itself to the ontologies used in the organization, mapping between agentís internal knowledge and systemís ontologies are defined for each agent. This mapping differs for different types of agents within the organization. For example, agents playing the role of information providers need a mapping between its local knowledge and the ontologies they support. On the other hand, agents playing the role of information requesters just need proper user interface definitions based on the ontologies they support. These mappings are documented using the extensible style sheet language for transformations (XSLT) [20] standard and stored for each agent.

Detailed Interaction Model. This model maps the interaction patterns identified in the analysis phase to the FIPA specifications. For this purpose, each message between the agents is rewritten using the FIPA ACL specification. The content of each message is also written in one of the content languages supported by FIPA such as FIPA-RDF.

At the end of the design phase, all of the concrete entities of the organization are ready to be implemented using any of the FIPA-compliant agent frameworks, since they totally conform the specifications of the FIPA.

3 Applying the SABPO Methodology

To illustrate how SABPO is applied to engineer a real system, we will use a part of the realistic B2B application as an example. This application is used by a developer company, which gathers required parts from different suppliers. Hence, the application should support the following functionalities:

- It should support the developer company users to define the desired partís knowledge
- It should find the potential suppliers of the desired parts.
- It should send a request to collect the information about the desired parts to the suppliers and shows the results to the developer companyís interested users.

Because of the space limitation, only analysis phase of the SAPBO is illustrated for the requirements listed above.

Role Model. ìDeveloper Companyî role and ìSupplierî role can be identified easily from the requirements. Following the Gaia methodology ìDeveloper Companyî role must have a ìpart definitionî activity, which helps the users to define the desired parts. A ìrequestî protocol must also be added to the ìDeveloper Companyî role to send the part request to the proper suppliers. On the other hand, ìSupplierî role must have a ìget part knowledgeî activity and an ìinformî protocol to return desired part knowledge to the ìDeveloper Companyî role.

However, there are some open questions such as how the "Developer Company" role will identify the right suppliers and how it will create the part knowledge that is understandable by suppliers. This is the point where "Directory Service Provider" and "Ontology Service Provider" roles come in to the stage. To locate the right supplier, "Developer Company" role must request this knowledge from the "Directory Service Provider" role since each supplier should advertise itself to the directory and the directory service provides such a service according to the FIPA abstract architecture specification.

To create understandable part knowledge, it is certain that this organization must have a global explicit ontology. Let's call this ontology as "part-ontology". "Supplier" role must support this ontology and advertise it to the directory as the supported ontology. "Developer Company" role must also support this ontology to create sharable part knowledge. As a summary, "Developer Company" role must use a "request" protocol to get the right suppliers that support the "part-ontology" from the directory service and each agent must have the standard "advertise" protocol to advertise itself (including the ontology it supports) to the "Directory Service Provider".

Interaction Model. Interaction Model In the interaction model, all protocols identified in the role model are defined using the Agent-UML [2]. Also, the general interaction mechanism is visualized using the collaboration diagram of the Agent-UML. Fig.3 shows the collaboration diagram of the role's interaction.

The information flow of the interaction in fig. 3 is given in detail below:

1. "Developer Company" requests the possible suppliers supporting the "part ontology" from the "Directory Service Provider".
2. "Directory Service Provider" informs the "Developer Company" by returning the possible suppliers.
3. "Developer Company" requests from the supplier for the desired part using the "part ontology".
4. Suppliers return the requested part information to the "Developer Company".

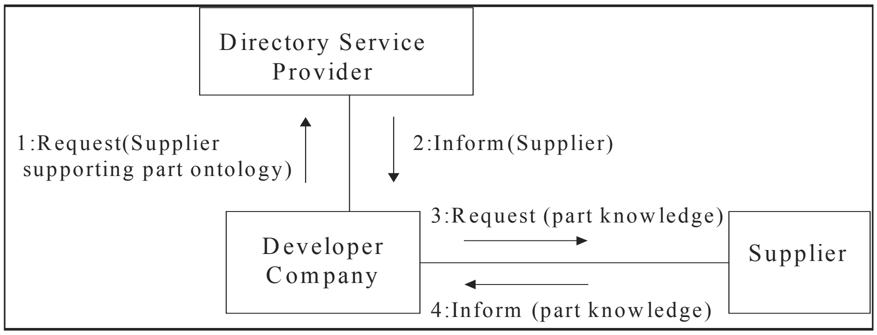

Directory Service Provider
1:Request(Supplier supporting part ontology)
2:Inform(Supplier)
3:Request (part knowledge)
Developer Company
Supplier
4:Inform (part knowledge)

Fig. 3. Collaboration diagram of the role's interaction

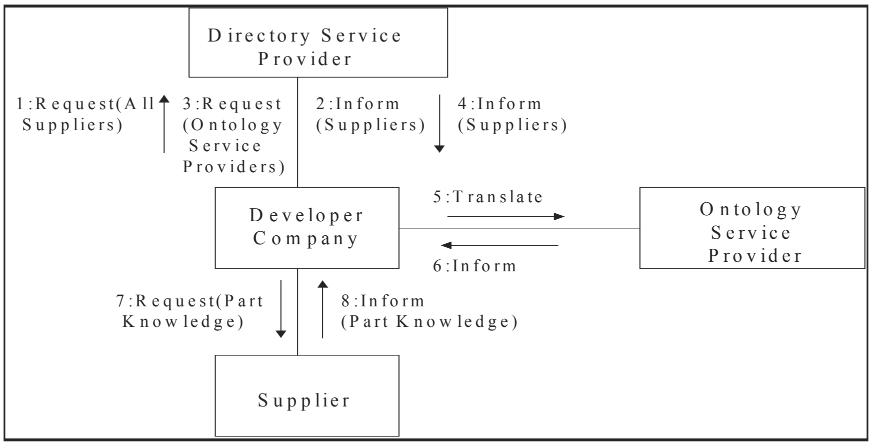

Directory Service Provider
1:Request(All Suppliers)
3:Request (Ontology Service Providers)
2:Inform (Suppliers)
4:Inform (Suppliers)
Developer Company
5:Translate
6:Inform
Ontology Service Provider
7:Request(Part Knowledge)
8:Inform (Part Knowledge)
Supplier

Fig. 4. Collaboration diagram of the role's interaction for a different situation

Now, we will change the situation and illustrate how SABPO handles different conditions using the FIPA specifications. Let us assume that there is another ontology in this domain supported by some suppliers and call this ontology as "part1 ontology". "Developer Company" supports only the "part ontology", but it also knows that there are some suppliers around supporting different ontologies. To handle this situation, there must be a relationship between these ontologies and the "Ontology Service Provider" should be able to perform the translation service between any two ontologies as defined in the FIPA Ontology Service Specification [10]. The collaboration diagram to handle the defined situation is shown in fig. 4.

Explanation of the information flow is given below:

1. "Developer Company" requests all suppliers in the domain from the "Directory Service Provider".
2. "Directory Service Provider" returns all possible suppliers.
3. "Developer Company" understands that there are some suppliers, which support "part1 ontology" and ask the "Directory Service Provider" if there is any "Ontology Service Provider", which provides a translation service between these ontologies.
4. "Directory Service Provider" returns the "Ontology Service Provider" which supports the translation service.
5. "Developer Company" requests for the translation of the query from "part ontology" to "part1 ontology".
6. "Ontology Service Provider" returns the translated query.
7. "Developer Company" requests for the part knowledge from suppliers using the proper ontology for each supplier.
8. Suppliers return the part knowledge.

This new situation, of course, creates new activities for each agent and new interaction protocols as shown in fig. 4.

As a summary, SABPO first defines the roles of the organization and their responsibilities in the analysis phase. These individual roles are accepted as the abstract agent element of the FIPA's abstract architecture. Then, it fully exploits the abstract architecture by taking the directory and the ontology services as roles and defining the interaction protocols between the roles to satisfy system's requirements.

4 Ontology Inception Pattern

MASs aim to model complex applications running on complex environments. Hence, MAS developers may always face new situations, which have not been defined in the FIPA specifications yet. If the solutions found for these new situations are general and applicable to any MAS facing similar situations, then these solutions should be added to the FIPA specifications. By this way, an extensible catalogue of solutions patterns will be formed as part of FIPA standards. This approach has been applied successfully by the software community in creating a pattern catalogue for solving recurring design problems in an extensible way [11]. We believe that pattern identification must be taken as one of the most important research direction of the agent community. In this section, a new interaction pattern is introduced for multi-agent systems and a documentation style is proposed to define the identified patterns.

4.1 Motivation and Interaction Mechanism

To document the MAS interaction patterns, we define the pattern from three different perspectives. First, motivation behind the pattern identification is introduced. Then, interaction mechanism of the participating agents is defined according to the FIPA standards. Finally, guidelines are defined to make the pattern FIPA-compliant.

Motivation. To understand the motivation behind the "Ontology Inception" pattern, let us look at the open environment from the ontological perspective and assume the following characteristics:

- There are a large number of domain dependent ontologies, which may change at run-time.
- At any time, new ontologies may be added to the MAS. Also, some existing ontologies may decided not to be used any more.

In such an environment, some questions easily come to mind. For example, one question is about how agents provide the information related with newly added ontologies. In the ontology service specification, OA only provides translation service, which may be used for this situation. But, any agent that supports different ontologies can join the organization at any time. Who will define the mappings between these new ontologies and existing ones? How will individual agents begin to support these new ontologies? It is clear that translation service is not enough to support such an environment. Hence, "Ontology Inception" is a solution pattern that makes agent organization adaptable to the open environment.

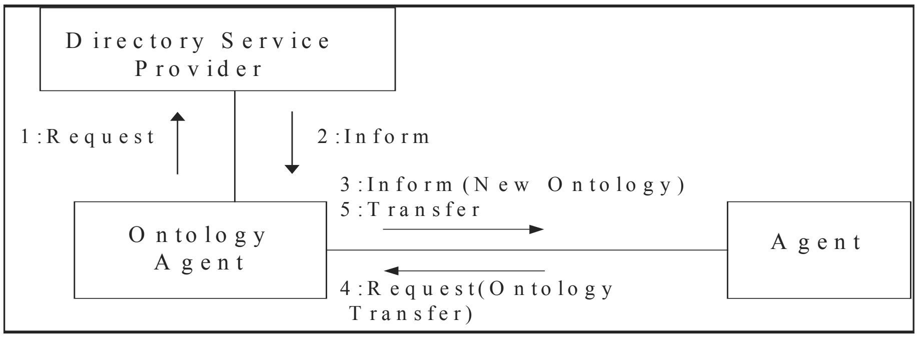


Fig. 5. Interaction mechanism of "ontology inception" pattern

Interaction Mechanism: Interaction mechanism defines the interaction protocol between the agents that collaborate to solve the problem at hand. Also, activities of the participating agents are identified and modeled as services using Gaia style. The interaction mechanism is illustrated in fig. 5 using a collaboration diagram.

Information flow between the participating agent is described below:

1. Ontology agent requests the directory facilitator agent to find the agent within the organization.
2. Directory facilitator sends the list of active agents to the ontology agent.
3. Ontology agent informs each agent that there is a new ontology in the system.
4. Interested agents send a request to the ontology agent for ontology transfer.
5. Ontology agent transfers the ontology to the interested agents.

It can be seen from the interactions that the agent role needs an activity to define the required mappings between the transferred ontology and its local knowledge. This activity is called as "localize" and defined as the service of the agent role. Also, ontology agent needs a new communication act to transfer a specific ontology to the interested agent. This new communication act is named as "transfer" and defined in Agent-UML.

Guidelines for FIPA Comformance. Guidelines for FIPA-compliance: The following guidelines are defined to conform the "Ontology Inception" pattern to the FIPA specification:

- A new service should be defined for the ontology agent to transfer the required ontology.
- FIPA-Meta-Ontology specification should be extended with required semantic to support the transfer service.
- FIPA standards committee should decide either to define a new communicative act for transfer protocol or using inform act inserting the required knowledge to the content of the message.

4.2 Run-Time Behaviour of the Ontology Inception Pattern

Sometimes it can be difficult to understand the internal behaviour of participating agents when they involve in a complex interaction pattern. "Ontology Inception" also requires a definition of how the participating agents perform a "localize" activity. The "localize" activity needs different behaviour for different types of agents. For example, an information provider type of agent implements it by defining a mapping between the transferred ontologies and its local knowledge. On the other hand, an information requester type of agent implements it to create user interfaces for the transferred ontology. Hence, sometimes it can be helpful to discuss the details of internal behaviour of the participating agent to make the pattern more understandable for the developers.

To transfer ontologies, first of all, it must be decided what representation to use for modeling the ontologies. In agent literature, description logic based languages [8], [12] are widely used for ontology representation. Also, there are some work which use UML for agent based systems, present an ontology representation language based on a subset of UML together with its associated Object Constraint Language (OCL) or propose mechanisms for reasoning on UML class diagrams using description logic [1] [6]. Similarly, we use UML for representing ontologies [7].

After deciding which ontology modeling language to use, the ontologies are represented in the selected ontology modeling language. Then, the represented ontologies are transferred by the ontology agent. Hence, developers have to decide how to import the ontologies in the content of FIPA ACL messages. One possible solution is to use FIPA RDF content language, since most of the ontology modeling languages are based on RDF and use RDF/XML as a syntax to transport the modeled ontologies.

Each participating agent can issue a transfer request for a specific ontology to the ontology agent, whenever needed. When the requested ontology is transferred from the ontology agent to the participating agent, the participating agent has to extract the content of the arrived FIPA ACL message and parse the incoming ontology. If the participating agent is an information provider type of agent, then it should visually show the ontology model and its local information model so that a mapping can be defined. This mapping will then be saved as part of the wrapping knowledge. If it is an interface type of agent, then it should generate a user interface so that the users can enter queries related with a specific ontology and can see the results.

5 Conclusion

In this paper, a new methodology, which uses organizational metaphor as the basis and integrates this metaphor to the FIPA standards, is introduced. In addition, a new interaction pattern, which is called as ìontology inceptionî, is defined. The proposed methodology and the "ontology inception" pattern have been used in the development of a multi-agent system infrastructure for software component market-place [7], which has all the basic the characteristics of open environments.

References

[1] Baclawski, K., Kokar, M.K., Kogut, P.A., Hart, L., Smith, J., Holmes, W.S., Letkowski, J. and Aromson, M.L. Extending UML to support ontology engineering for the semantic web. The Unified Modeling Language, Modeling Languages, Concepts and Tools, 4th International Conference, Toronto, Canada, LNCS 2185 Springer, (2001) 342-360.

[2] Bauer, B., M¸ller, J.P. and Odell, J. Agent UML:A formalism for specifying multiagent interaction. Agent Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer-Verlag Berlin (2001) 91-103.

[3] Bellifemine, F., Poggi, A., and Rimassa, G. Developing multi-agent systems with a FIPA-compliant agent framework. Software Practice. and Experience, (2001) 31:103-128.

[4] Bergenti, F. On Agentware:Ruminations on why we should use agents. In Petta, P., Tolksdorf, R., Zambonelli, F. and Ossowski, S. (eds). Worksop Notes of the Third International Workshop Engineering Societies in the Agents World (ESAWí02), Madrid, Spain (2002) 9-20.

[5] Bernon C., Gleizes, M., Peyruqueou, S., and Picard, G. ADELFE, a methodology for adaptive multi-agent systems. In Petta, P., Tolksdorf, R., Zambonelli, F. and Ossowski, S. (eds). Worksop Notes of the Third International Workshop Engineering Societies in the Agents World (ESAWí02), Madrid, Spain (2002) 21-34.

[6] Cranefield, S., Haustein, S., and Purvis, M. UML-based ontology modelling for software agents. Proceedings of the Workshop on Ontologies in Agent Systems held at the 5th International Conference on Autonomous Agents, Montreal Canada (2001). Available on line at http://cis.otago.ac.nz/OASWorkshop.

[7] Erdur, R. C. and Dikenelli, O. A multi-agent system infrastructure for software component market-place: an ontological perspective, ACM Sigmod Record, Vol:31, Number:1, (2002) 55-60.

[8] Fensel, D., Horrocks, I., Van Harmelen, F., Decker, S., Erdmann, M. and Klein, M. OIL in a nutshell. In the proceedings of the workshop on applications of ontologies and problem solving methods, 14th European Conference on Artificial Intelligence, Germany (2000).

[9] FIPA(a). FIPA XC00001I: FIPA abstract architecture specification. http://www.fipa.org.

[10] FIPA(b). FIPA XC00086C: FIPA ontology service specification. http://www.fipa.org.

[11] Gamma, E., Helm, R., Johnson, R. and Vlissides J. Desing patterns. Addison Wesley, Reading (MA) (1995).

[12] Hendler, J. and McGuiness, D. The DARPA agent markup language. IEEE Intelligent Systems, 15, No.6, (2000) 67-73.

[13] Iglesias, C., Garijo, M. and Gonzales, J. A survey of agent-oriented methodologies. In A.S. Rao J.P. Muller, M.P.Singh, editor, Intelligent Agents IV (ATAL98), LNAI Springer- Verlag (1999).

[14] Kendall,E.A. Agent software engineering with role modeling. In Proceedings of the 1st International Workshop on Agent Oriented Software Engineering, volume 1957 of LNCS. Springer Verlag (2000).

[15] Kinny, D. and Georgeff, M. A methodology and modeling technique for systems of bdi agents. In Workshop on Modeling Autonomous Agents in a Multi-agent world, LNAI 1038, Springer Verlag (2000) 56-71.

[16] Ominici, A. SODA: Societies and infrastructures in analysis and design of agent-based systems. In Ciancarini, P. and Wooldridge, M. (eds). Proc. 1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000), Limerick, Ireland, June, 2000. Volume 1957 of LNCS, Springer-Verlag Berlin (2001).

[17] Poslad, S., Buckle P., and Hadingham, R. The FIPA-OS agent platform: open source for open standards. In the Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-agents, U.K. (2000). Available at
http://www.fipa.org/resources/byyear.html.

[18] Wood, M., DeLoach, S.A. and Sparkman, C. Multiagent system engineering. International Journal of Software Engineering and Knowledge Engineering, Vol:11 No:3 (2001) 231-258.

[19] Wooldridge, M., Jennings, N.R. and Kinny, D. The Gaia Methodology for agent oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems, (2000) 3(3):285-312.

[20] XSLT specification, http://www.w3.org/TR/xslt.

[21] Zambonelli, F. and Parunak, V. D. Signs of a revolution in computer science and software engineering. In Petta, P., Tolksdorf, R., Zambonelli, F. and Ossowski, S. (eds). Worksop Notes of the Third International Workshop Engineering Societies in the Agents World (ESAWí02), Madrid, Spain (2002) 265-279.

[22] Zambonelli, F., Jennings, N., and Wooldridge, M. Organisational rules as an abstraction for the analysis and design of multi-agent systems. Int. Journal on Software Engineering and Knowledge Engineering. 11(3) (2001) 303-328.

Modelling a Multi-agent System Environment

Sehl Mellouli, Guy W. Mineau, and Daniel Pascot

Laval University, Computer Science Department
G1V 7P4,Quebec, Canada
{sehl.mellouli,guy.mineau}@ift.ulaval.ca
daniel.pascot@sio.ulaval.ca
http://www.ift.ulaval.ca/~lic

Abstract. A multi-agent System (MAS) can be viewed as a software system evolving in some environment, with which it has to interact. During the MAS life cycle, many situations may occur, situations that are considered to be critical and that would have an effect on the MAS, prompting it to quickly adjust to this new situation. A design methodology of a MAS should help the designer to represent information about a changing environment and its effects on the MAS, an aspect of the modelling task which is currently lacking from agent design methodologies. We propose to add two new diagrams: an environment diagram and an agent diagram, to MAS modelling methodologies. These diagrams will conceptualise the impact of the environment on the structure of the MAS, and therefore should guide the development of the actual implementation of the MAS.

1 Introduction

A multi-agent system (MAS) consists of a number of agents that operate together in order to achieve some goals. It can be viewed as an *agent* organization (by analogy with *human* organization) or in other words, as some artificial society or organization [11]. In fact, like human organizations, a MAS evolves in a certain environment, has goals to achieve and its agents operate together to achieve these goals. So we feel that their design should be somewhat inspired from that of human organizations (based on organization theories) [6]. From the study of organization theories, we find that organization structures are viewed as metaphors (as summarized in Morgan [7]) that can be applied for the study of many sorts of organizations [2]. It is important to notice from the start that human organizations are complex and most of the time, they need more than one metaphor to be accurately described, or need some variant of a metaphor. We have chosen the four metaphors that seem to be applicable to multi-agent systems: the *machine*, *organism*, *brain* and *political* metaphors. Based on these metaphors, we have identified five key dimensions to design a human organization that we adapt to the design of an agent organization [6]. These dimensions are: the nature of the environment, the tasks to be performed, the control, communication and collaboration relationships between agents.

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 227-240, 2003.

Looking at agent-oriented methodologies such as Gaia [11], AUML [8] and MAS-CommonKADS [8], we find that they do not integrate any mechanism to explicitly model the environment, its evolution and its impact on the structure of the MAS [6]. They have no formal representation of the specifications of the MAS, and therefore, there is no way to implement some early validation of the behaviour of the MAS under design.

We define the environment as a set of situations that can occur during the organization life cycle, and where each situation has an impact on the structure of the organization and on its meaningful behaviour. We feel that the modelling of the environment and its impact on the organization structure of the MAS is vital to the usefulness of the system since it challenges the adaptability of the system.

We propose to add two diagrams to MAS modelling methodologies in order to conceptualise and represent the environment and its impact on the organizational structure of the MAS. These diagrams are the environment diagram and the agent diagram.

The environment diagram. It represents a state transition diagram between the different situations that the environment can undergo. A change in a situation will affect most probably the roles that the agents play and their relationships. This diagram is needed in the modelling of this dependency between environmental situations and a MAS. We define loosely the environment as being the set of the different situations that condition the behaviour of the system, and the possible sequence of occurrences of these situations. Through domain analysis and background knowledge, we determine what situations are likely to have a tremendous impact on the organization of the MAS and, for obvious complexity reasons, we restrict ourselves to them.

The agent diagram. Defined in relation to the environment diagram, it represents the structure of the agents (roles) and their relationships (control, collaboration and communication relationships), and how the environmental situations affect them.

This paper is organized as follows. In section 2, we present the limits of the Gaia [11], AUML [8] and MAS-CommonKADS [3] methodologies with regard to the modelling of the environment. Section 3 presents the environment diagram and Section 4 presents the agent diagram based on some soccer game example. Section 5 concludes.

2 Limits of Agent Oriented Methodologies

In this section, we focus our study on three methodologies: Gaia [11], AUML [8] and MAS-CommonKADS [3]. We have chosen these methodology since Gaia is a recent one, AUML is based on UML which is a standard in object oriented development, and MAS-CommonKADS which is based on CommonKADS used to develop expert systems.We model some strategic tactics of a soccer team using Gaia, AUML and MAS-CommonKADS. We centre our argumentation around the modelling of the environment (a set of situations that impacts the organization structure of the MAS) and show how these methodologies do not allow the designing of a MAS that could readjust its overall behaviour to an evolving environment. We also show how the five dimensions that we identified from [6] (environment, task, control, communication and collaboration) as being important to the modelling of any organization are taken into account by these methodologies. From that presentation we aim at showing that no provision was made to explicitly model the environment under these three methodologies.

2.1 The Gaia Methodology

Gaia [11] is based on three phases. The first phase is requirements statement; the second phase is analysis and the third phase is design. The first phase, requirements, is independent from any implementation. It corresponds to use cases in object oriented design. The second phase, analysis, specifies the different roles to be developed in the software system, and their interactions. It is composed of the role model and the interaction model. The former identifies the key roles that must be undertaken by agents in the system. The latter identifies the interactions between roles according to a set of protocol definitions. The third phase, design, defines the different agent types and instances (the agent model), the different services offered by each agent (service model) and the communication between the agents (acquaintance model). Each agent can be associated with a set of roles.

The basic notion in Gaia is the role. It is very important to precisely identify the key roles of the MAS since all models produced by Gaia are based on them. When modelling a soccer team, the key roles are : Goal Keeper (GK), Defender (DF), Mid-Fielder (MF) and Striker (ST). In Gaia, each role is described by a schema (see Table 1). In this schema we:

- Give a textual description of the role: the role of the defender is to stop adverse forward players,
- Determine the different protocols and activities that will be played by the role: the defender has the activity to stop adverse players (stopPlayers) and must use protocols to guide his team mate in the overall defence strategy of the team (guideDefense),
- Determine the permissions associated to the roles. A player always has the possibility to kick the ball (kickBall),
- Define the responsibilities of the role that are composed of liveness and safety responsibilities. The liveness responsibilities are the functionalities of the role. The defender must stop adverse players (stopPlayers) and guide his team mate in defence (guideDefense). To fulfil its role, the defender must warranty that no adverse player scores a goal in his teamís net. This constitutes the safety responsibilities.

According to the dimensions identified in organization theories (environment, task, control, communication and collaboration), Gaia does not deal with the environment, control and communication relationships. In fact, if we need to represent a situation where a player is injured, there is no way to do it. Also, if a defender must become a middle fielder for tactical reasons, there is no way to represent the evolution of the role of a player during the game. Each agent has a specific role determined during the design phase and cannot evolve over time. There are no mechanisms (at least during the design phase) to foresee and model this evolution. Also, if we need to represent that an agent controls another agent, there is no mechanism to do it.

Table 1. Schema for role Defender

Role schema:	Defender
Description:	Stops adverse forward players
Protocols and Activity	stopPlayers, guideDefense.
Permissions	kickBall
Responsibilities	
Liveness:	stopPlayers, guideDefense
Safety:	No adverse player scores

2.2 AUML

AUML is an extension of UML for modeling MAS [8]. It identifies agents by a class diagram describing the roles of each agent [1]. An agent class is identified by its name, its different roles, a state-description, actions, methods, capabilities, constraints and agent-head-automata-name. For the soccer example, if there is an agent player called A6 whose role is a DF (Defender), then the agent class (part of it) is described as follows (see Figure 1).

According to the dimensions identified in organization theories, AUML deals only with communication, collaboration and tasks (via roles) but does not provide any way to deal with the environment and control relationships. In fact, if we need to represent a situation where a player is injured, there is no way to do it. Also, if a defender must become a midfielder for tactical reasons, there is no way to represent the evolution of the role of a player during the game. Each agent has a specific role determined during the design phase and cannot evolve over time. Also, if we need to represent that an agent controls another agent, there is no mechanism to do it. But we note that in [10], there is an attention paid to the important role played by the environment in designing a MAS. Nevertheless, there is still no way to represent the environment within AUML and no control relationships between agents in the agent class diagram.

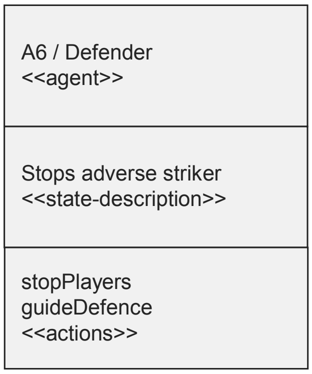


Fig. 1. A part of the agent class as defined in AUML

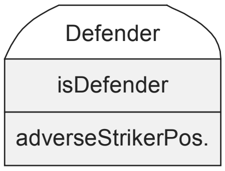


Fig. 2. MAS-CommonKADS central defender agent class

2.3 MAS-CommonKADS

This methodology is an extension of the CommonKADS methodology [9] for agent systems using techniques of Object Oriented methodologies. It is based on the development of seven models: agent model, task model, knowledge model, organization model, coordination model, communication model and design model. Theses models are constructed in three stages: conceptualization, analysis and design. The first stage helps to understand the work of the future system by developing use cases and Message Sequence Charts (MSC). The analysis stage of the method is the production of all models except the design model, which will be produced at a later stage.

The agent model is constituted by agent classes. Each agent class is described by three parts. The first part is the role of the agent. The second part is the mental state and internal attributes of the agent, such as its beliefs, goals or plans. The third part is the external attributes of the agent such as the sensors and effectors of the agent. For the soccer game, the agent class for defender agents is presented as follows (see Figure 2) :

The role of the agent is defender. It must be aware of it (part of its beliefs) and must know the position of adverse strikers (through external attributes) to counter any attack.

According to the dimensions identified from organization theory, MAS-CommonKADS does not deal with the environment and the control relationship between agents. If the defender has to change its role to a midfielder for tactical reason, there is no way in MAS-CommonKADS to model that information. Also there is no way to represent the environment in which a MAS evolves, and therefore, no provision is made to formally represent agent behaviour in relation to the environment, allowing possible failure situations to subsequently occur as a result of this under specification.

Looking at agent oriented methodologies [3, 8, 11], we find that they do not take into account the environment and its impact on the organization structure of the MAS. There is no formal way to include the description of the environment into the conceptualization of a MAS using these three methods. Since it has an influence on the structure of the MAS, we introduce, in the next section, the environment diagram that can be added to MAS modelling methodologies. The environment could affect the roles or the tasks played by the agents, the communication between two agents, the control of one agent on another or the collaboration between agents. From this, we propose an agent diagram where the agent is identified by its name, the roles it has to play and where control, communication and collaboration relationships between agents are also represented. We will also propose to add a control relation between agents, in the agent diagram, since the environment diagram may impact on it.

3 The Environment Diagram

A Multi-Agent System (MAS) is designed to achieve specific goals. A MAS is placed in an environment with which it interacts. The MAS must decide what to do and develop a strategy in order to achieve its assigned goals. For this, the MAS must have a representation or a model of the environment in which it evolves. We now ask the question: *how can the environment be modelled?*

A part of this answer is given by McCarthy in its situation calculus [5]. For McCarthy, the environment (or world) is composed of situations. A situation is the complete state of the world at an instant of time: a snapshot of the world at some instant. Each situation can be followed by an event that leads to a new situation. The situation calculus allows to infer something about future situations. Let us define sí = result(e,s) where sí is a situation that can be reached from situation s when event e occurs. It is specifically designed for representing dynamically changing worlds.

Moreover, the SODA [9] and ADELF [2] methodologies consider the environment in their process to design MAS; however, no formal representation is defined to model the environment. This is what we propose in this work. Based on McCarthy theory, and since agent oriented methodologies as presented in section 2 lack in this regard, we propose to add to these methodologies an environment diagram that models different situations of the environment and their transitions. We define the environment as a set of situations and transitions between these situations. Each situation is described by a certain number of *critical* parameters. These situations are considered to be important in the design of a MAS since they could for example affect its organization structure or prevent it from achieving its goals. The transitions between situations are defined as events. In the soccer example, many critical parameters can be identified such as: the remaining time, the score difference or the team deployment tactic. The change of the value of one of these parameters could affect the roles played by the agents and so the organization structure of the MAS. So it is important to identify these critical parameters, and describe various situations of the environment that should have an impact on the MAS behaviour.

Each situation within the environment is described by a set of valued parameters. The set of parameters describing a situation s_i is P_i. Each parameter p_i of P_i has a single[1] value v_i. There could exist two situations s_i and s_j where $P_i \subseteq P_j$. If also, the value v_j (of an element p_j) is equal the value v_i of the element p_j in P_i ($P_i \subseteq P_j$), we will say that s_i is a generalization of s_j, and s_j is a specialization of s_i. We conclude that if a situation s_k leads to situation s_j, so it leads to situation s_i; and if the situation s_i leads to a situation s_k, then the situation s_j leads to the situation s_k.

Formally, the environment structure S is defined as S = <I, W, E, R, F> where I is the set of initial situations, W is the set of all situations ($I \subseteq W$), E is the set of all possible events, R is a relation that represents transitions between set of situations with

[1] We can extend the formalism for multi-valued parameter.

$R \subseteq \wp(W)$ x E x $\wp(W)$ where $\wp(W)$ is the power set of W. R is a triplet (S_i, e, S_j) where S_i and S_j are in $\wp(W)$ and e is in E. The triplet (S_i, e, S_j) means that from any element of S_i, if the event e occurs, the system will subsequently be in situation S_j. All the elements of S_i, respectively S_j, are related to each other by the relation of generalization-specialization as defined above. The set F is the set of final situations that can be reached ($F \subseteq W$). There are no possible situations after the elements of F, i.e., no triplet of R has the structure (f,e,fi) for whatever e in E and fi in W. The situations and their transitions define the environment diagram [6] which is a state (situation) transition diagram. It is a graphical representation of R such that it is a graph where nodes represent elements in W and where an arc e between two situations si and sj in W exists if there exists two sets $S_1 \in \wp(W)$ and $S_2 \in \wp(W)$ where $s_i \in S_1$, $s_j \in S_2$ and < S_1, e, S_2> ∈ R.

In the soccer example, we have identified three parameters (for simplicity reasons) which, together, define situations that may impact the roles of agents. These parameters are: the Score Difference (SD), the Remaining Time (RT) and the Deployment TactiC (TC). If SD = +*v*, for $v \in N^*$, it means that the team is leading by *v* goals. If SD = -*v*, then it is loosing by *v* goals. The RT parameter takes values of the form: op t, where $t \in N^*$, and op ∈ {=, <, >, ≤, ≥}. Finally, TC is defined as = n_1-n_2-n_3, such that $n_1+n_2+n_3$ = number of players on the field (for the same team), where n_1 is the number of defenders, n_2 is number of midfielders and n_3 is the number of strikers.

An environment diagram such as what we propose, i.e., a situation transition diagram, would help to formally represent the characteristics of the environment. In each situation, we give a name that identifies the situation, and values to the different parameters characterizing it. For the soccer example, we restrict our study to only eight situations (see Figure 3). Among these situations, there are special ones s_0, s_1, s_3, s_5 and s_6. s_0 is the initial situation identified by a bold rectangle. It is the first situation that could occur and represents the environment when the game starts. s_1, s_3, s_5 and s_6 are final situations; there are no possible situations after them.

Each situation is described by the three parameters SD, RT and TC. For example the situation s_0 represents a situation where the score difference (SD = 0) is equal to 0, where there remain 90 minutes of playtime (RT = 90), and where the adopted tactic is 4-4-2 (TC = 4-4-2).

According to the definition of the environment <I, W, E, R, F>, the set I = {s_0}, W = {s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7}, E = {box_goal, scores_goal, time_<_25, injured_player, recovered_player), R = {({s_0}, box_goal, {s_1}), ({s_0}, scores_goal, {s_2}), ({s_0}, time_<_25, {s_4}), ({s_2}, box_goal, {s_5}), ({s_4}, scores_goal, {s_3}), ({s_4}, injured_player, {s_7}), ({s_7}, recovered_player, {s_6})} and F = {s_1, s_3, s_5, s_6}.

Once the situations are identified, it is interesting to see their impact on the structure of the MAS, more specifically on the roles played by the individual agents and their relationships. For this purpose, we need to use an agent diagram [6] to represent agents, and to describe their roles and relationships. This agent diagram is drawn early on when designing a MAS. This kind of agent diagram is not defined in any of the mentioned methodologies: Gaia, AUML and MAS-CommonKADS. This is what we present in the next section.

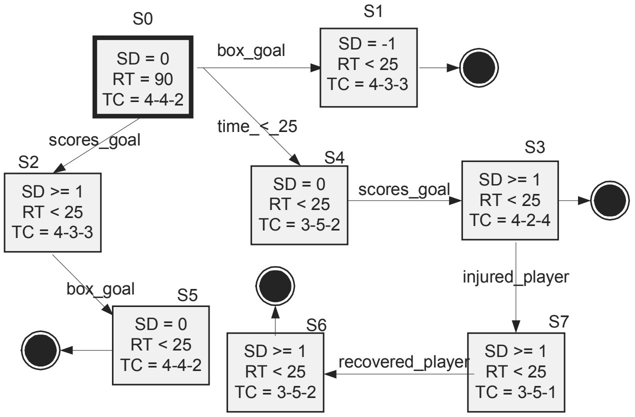


Fig. 3. The environment diagram of a soccer game

4 The Agent Diagram

The environment diagram is composed of situations that could have an impact on the organization structure of the MAS according to the roles played by the agents and their relationships (control and communication). Some of these situations may lead the MAS to a failure situation. So, we propose to define a diagram that would show the impact of any situation of the environment on the organization structure of the MAS; this is the agent diagram.

In the agent diagram, each identified agent is specified by its name and roles. Since control and communication relations[2] are at the centre of a MAS architecture, it is vital for the modeler to represent them. Their explicit representation favours knowledge elicitation (more semantics is added to the agent diagram); their formal representation favours automatic validation of the system. Furthermore, the maintenance of the system will be facilitated by such an explicit representation of the system, and so will the automatic generation of the code. Each agent diagram (or subpart of it) is attached to a situation, the one identified by the label *Situation name* in the agent diagram. To see, how the agent diagram is related to the environment, we now continue with the soccer example. In the initial state s_0, the team is organized according to a 4-4-2 deployment tactic: four defenders, four midfielders and two strikers. Any player can communicate with any other player. Of course the volume of communication is more abundant between players who share the same role. The agent diagram associated with situation s_0 is represented in Figure 4.

In Figure 4, we have omitted to represent communication and control relationships between roles where the communication is not abundant. For example, we suppose that since agent A8 controls the midfielders team mates, he is the only responsible agent for this task, and so there is no control no communication relation necessary between A7, A6 and A9.

[2] For now, we will consider collaboration as a kind of communication.

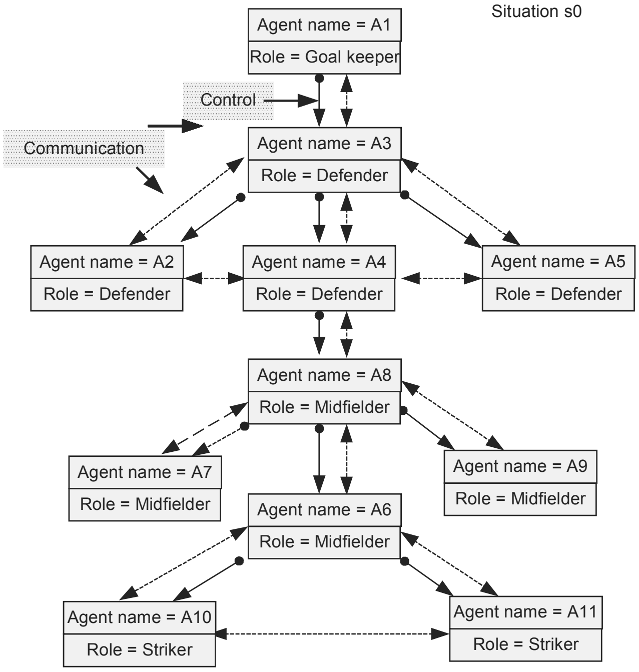


Fig. 4. The agent diagram D_1 associated with situation s_0

In situation s_2, the team is organized along a 4-3-3 tactic. The A6 player changes its role to become a striker. The volume of communication between him and the other strikers will increase and decrease between him and other middle-fielders. Figure 5 represents how the MAS structure changes with situation s_2 where the role of A6 has changed to become a striker (the changes are in bold). Of course one could imagine that the drawing of Figure 5 could be done incrementally from that of Figure 4 in a MAS case tool.

By comparing the agent diagrams associated with sequentially occurring situations, the information on how the architecture of the MAS will be affected by these changes is depicted. The agent diagram structure is defined as $D_a = \{A, C, R_a\}$ where A is the set of agent instances, C is the set of relationships that can be defined between agents, and R_a is a relation that determines the relationships between agents; $R_a \subseteq A \times C \times A$. Each agent diagram is related to a particular situation in the environment diagram. We define D as a set of various agent diagrams D_a, so $D = \{D_a\}$.We define a relation $R_{sa} \subseteq W \times D$ that relates a situation W in the environment diagram to some agent diagram. In the soccer game example, according to Figure 5, (s_2, D_2) belongs to R_{sa}.

The agent diagram D_2 is defined as $D_2 = \{A, C, R\}$ where A = {A4, A6, A7, A8, A9, A10, A11}, C = {control, communication} and R_a = {(A4, control, A8), (A4, communication, A8), (A8, communication, A4), (A8, control, A7), (A8, communication, A7), (A7, communication, A8), (A8, control, A9), (A8, communication, A9), (A9, communication, A8), (A8, control, A10), (A8, communication, A10), (A10, communication, A8), (A8, control, A6), (A8, communication, A6), (A6, communication, A8), (A8, control, A11), (A8, communication, A11), (A11, communication, A8), (A6, communication, A10), (A10, communication, A6), (A6, communication, A11), (A11, communication, A6), (A10, communication, A11), (A11, communication, A10)}.

5 Applications

The environment and agent diagrams are used to design multi-agent systems where the environment is dynamic (evolves over time) and where the organization structure of the MAS evolves over time where the roles of agents and their relations change according to the environment evolution. Typical domain applications could be computer games, military applications or Internet applications. In what follows, we give an example of a military application.

5.1 The Military Application

We present the example of a group of five F-16 fighters piloted by automated software agents, that have to attack and destroy a nuclear plant. This is a highly unpredictable environment since many troublesome situations may occur such as an interception attempt from enemy fighters or from the enemyís air defense system. For each situation, the fighters must reorganize themselves, as they may have to regroup in smaller squadrons to engage the enemy. The MAS guiding the attack must quickly react to any new situation and try to reach its target as much as possible, or retreat. There is much information that must be shared at all times between the fighters of the squadron. It is important to identify this information, and describe various states of the environment that should have an impact on the MAS behavior. The useful information to the fighters in their various tasks are, among others:

- The position of the target: each agent must know how far the plant is and where it is located according to the position of the group.
- Position within the group: each fighter has a position within the squadron, that depends on the actual formation.
- The position of enemy fighters: knowing this position, the MAS can (re)organize itself to avoid an attack and to defend its unity. Counter-attack could also be organized.

To see how the agent diagram is related to the environment diagram, we now continue with our example. The squadron is composed of a head pilot that gives orders to the other fighters, of an auxiliary head pilot that can replace the head pilot in case of problems, and of three additional fighters. We find three major roles: head pilot, auxiliary head pilot and fighter. Two agents have two roles: head (or auxiliary head) squadron, and fighter. Three agents only act as fighters. All agents must probe the environment to make sure that they can achieve their goal. On the second and third states of the environment diagram of Figure 6, we can observe that some (critical) parameter values change as a result of some events. In each case, the MAS must be reorganized to overcome the two related situations. In the second state, the auxiliary head pilot replaces the head pilot. This induces the designation of a new auxiliary head. For the third state, the squadron must be split into two groups. One group will attack the enemy fighters while the other, if help is not needed by their teammates, will proceed to their target. Each environment state, being described by high impact parameters, plays a direct role in the (re)organization of the MAS. The agents play some roles in each state and we need some mechanism to model the change in agent roles. For this, we assume that we have a complete agent diagram of the MAS (not presented here for lack of space) as described in relation to the initial state of the environment diagram.

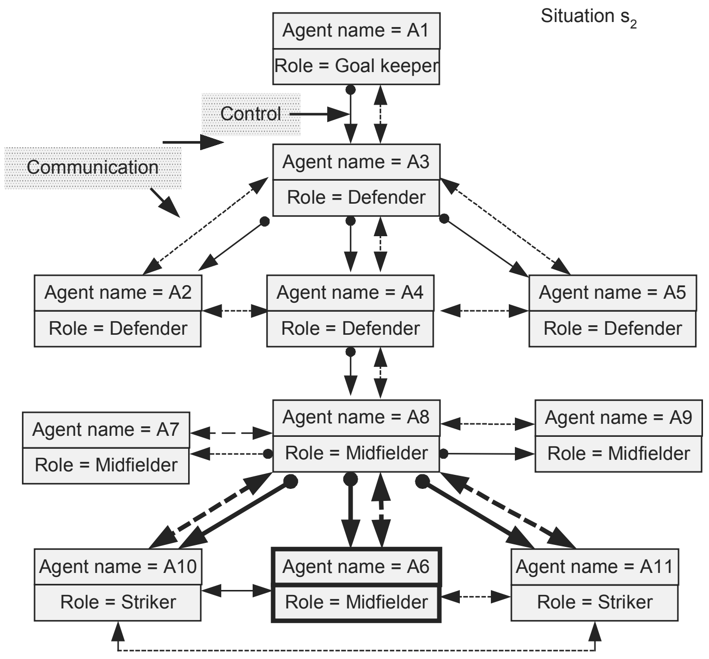


Fig. 5. The agent diagram D_2 associated with situation s_2: the role of A6 has changed and so have the relationships with the neighbouring agents

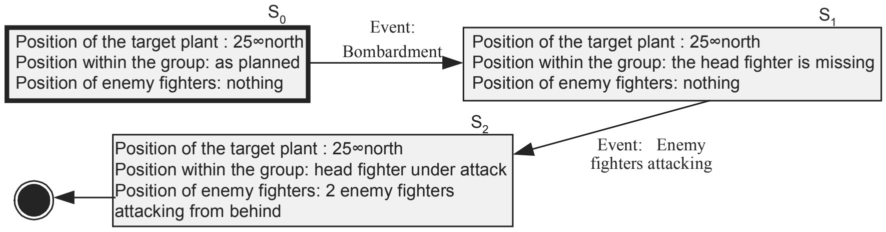

S_0
Position of the target plant : 25∞north
Position within the group: as planned
Position of enemy fighters: nothing
Event:
Bombardment
S_1
Position of the target plant : 25∞north
Position within the group: the head fighter is missing
Position of enemy fighters: nothing
S_2
Position of the target plant : 25∞north
Position within the group: head fighter under attack
Position of enemy fighters: 2 enemy fighters
attacking from behind
Event: Enemy
fighters attacking

Fig. 6. Environment diagram of the airplane attack

For example, a part of the agent diagram for the first environment state of our example is given in Figure 7.

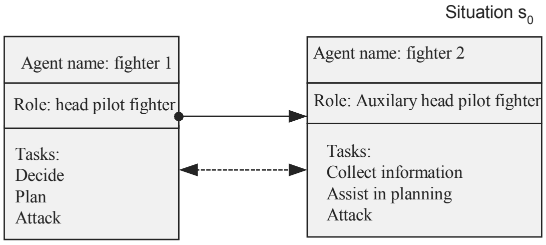


Fig. 7. Part of the agent diagram at state s_0

A part of the agent diagram for the second state is shown in Figure 8.

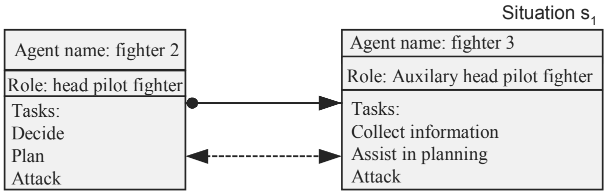


Fig. 8. Part of the agent diagram after head pilot is shot down

To summarize, a MAS has goals to achieve and acts in an environment with which it is in perpetual interaction. The behavior of the system is affected by the evolution of the environment over time. We propose to model each state of the environment as a set of critical parameters identified in the design phase. A changing in value of one of these parameters will lead to a new state of the environment. These parameters are critical on the correct behavior of the system. To represent the evolution of the environment over time, we propose an environment diagram which is a state transition diagram. Furthermore, the evolution of the environment should have an effect on the behavior of the system. So, we propose to attach to each environmental state, a subpart of the agent diagram that shows the impact of this state on the organization of the MAS.

6 Conclusion and Future Work

Based on organization theories, we have identified five key dimensions that must be considered when modelling some organization, either human or software-based: the environment, the nature of the tasks to be performed, the communication, control and collaboration links between the different entities (agents) of the system. Through an analysis of the three most popular MAS modelling methodologies: Gaia, AUML and MAS-CommonKADS, we could pinpoint that all of them do not provide diagrams for the explicit modelling of the environment in which the MAS will evolve. If this environment is dynamic, the MAS may have some instable behaviour.

Therefore, we propose two diagrams to add to agent oriented modelling methodologies: the *environment diagram* and the *agent diagram.* The environment diagram is a state transition diagram, and connects to agent diagrams, representing the impact of each state on the organization structure of the MAS. The agent diagram states how the MAS is organized according to the roles and tasks accomplished by agents. Our two diagrams are independent from any of the methodologies and can be added to any of them as extensions.

As future work, many issues are still opened for investigation. Since the environment is unpredictable, we are not able to predict which state will occur next other than by using a probabilistic approach in the representation of the situation transition diagram. So, we can evaluate, with a certain probability, whether the MAS will well function or not. It is also important to represent the beliefs of the agents within the agent model pertaining to the information given in each environmental situation, since the information the agent has is important to evaluate whether the MAS will well function. For now, our main objective is to provide a simple graphical model to the designer so that the environment and its impact on the MAS can easily be captured, and that it could be translated into a system of logic whose model checking procedure would provide assistance to the designer by guiding the modelling process and detect situations that could MAS to a failure to achieve its goals. Cost reduction and reliability issues of MAS development (specifically when dealing with dynamic environments) are two of the main concerns driving our research efforts.

References

[1] Bauer., B. UML Class Diagrams Revisited in the Context of Agent-Based Systems. In the Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada, May 28- June 01. 2001. pp 1-8.

[2] Bernon., C., Gleizes., G., Peyruqueou., S., Picard., G. ADELF, a Methodlogy for Adaptive Multi-Agent Systems Engineering. Workshop Notes of the Third International Workshop Engineering Societies in the agents world, 16-17 septembre 2002, madrid, spain, pp. 21-34.

[3] Carlsen., S., and Gjersvik., R. Organizational Metaphors as Lenses for Analyzing Workflow Technology. In proceedings of the international ACM SIG GROUP conference on Supporting Group Work: the Integration Challenge, Phoenis, AZ USA, November 16-19, 1997.

[4] Iglesias., C. A., and Garijo., M., and Gonzales., J. C., and Velasco., R. Analysis and Design of Multi-Agent Systems using MAS-CommonKADS. In proceedings of the AAAIí97 Workshop on agent Theories, Architectures and Languages, Providence, USA, July, 1997.

[5] McCarthy., J. Situation Calculus With Concurrent Events and Narrative. http://www-formal.stanford.edu/jmc/narrative/narrative.html, 2000.

[6] Mellouli., S., Mineau., G., and Pascot., D. The Integrated Modeling of Multi-agent Systems and their Environment. In Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems 2002 (AAMAS 2002), 15-19 July 2002, Bologna, Italy. pp 507-508.

[7] Morgan., G. Images of Organization. SAGE Publication. 1997. ISBN: 0-7619-0634-7.

[8] Odell., J., Van Dyke Parunak., H., and Bauer., Bernhard. Extending UML for Agents. Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, Gerd Wagner, Yves Lesperance and Eric Yu eds., Austin, Tx, pp 3-17, AOIS Workshop at AAAI 2000.

[9] Omicini., A. SODA : Societies and Infrastructures in the Analysis and Design of Agent-based Systems. Agent Oriented Software Engineering Workshop, 10 June 2000, Limerick, Ireland, pp 185-193.

[10] Van Dyke Parunak., H., Odell., James. Representing Social Structures in UML. Proceeding of the Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001) at the 5th International Conference on Autonomous Agents, pp 17-24, Montreal, Cananda, May, 2001.

[11] Wooldridge., M., and Jennings., N. R., and Kinny., D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (3).

Towards a Methodology for Coordination Mechanism Selection in Open Systems

Simon Miles[1], Mike Joy[2], and Michael Luck[1]

[1] Department of Electronics and Computer Science
University of Southampton, Southampton SO17 1BJ, United Kingdom
{sm,mml}@ecs.soton.ac.uk
[2] Department of Computer Science
University of Warwick, Coventry CV4 7AL, United Kingdom
msj@dcs.warwick.ac.uk

Abstract. Agent-oriented software engineering (AOSE) is a promising approach to developing applications for dynamic open systems. If well developed, these applications can be *opportunistic*, taking advantage of services implemented by other developers at appropriate times. However, methodologies are needed to aid the development of systems that are both flexible enough to be opportunistic and tightly defined by the application requirements. In this paper, we investigate how developers can choose the coordination mechanisms of agents so that the agents will best fulfil application requirements in an open system.

1 Introduction

Connecting together software environments (subsystems) to form dynamic open systems offers applications the potential to make use of the most up-to-date functionality in any of those environments at any time. That functionality can then be extended by adding more environments and adding new services to the existing environments. Now, underpinning much of the *raison d'etre* for multi-agent systems is that they are a highly appropriate technology for applications that run in dynamic open systems. By reacting to changes in the system state and making appropriate decisions, agents can, if well designed, be *opportunistic*. Opportunistic agents attempt to take advantage of the best functionality available, regardless of whether those services have been implemented by the developer of the agents or provided by another. In order to ensure that an application can be judged to be opportunistic, the design should be tightly defined by the application requirements, i.e. the design should have *justification*.

How well an agent can take advantage of the functionality available in the system, and the amount of control it has over activities within the system depends on the *coordination mechanisms* it uses. A coordination mechanism is a software process employed by an agent to ensure other agents behave as it would find most useful, usually making use of communication between the agents. Different coordination mechanisms provide different functionality but also place different demands on the system. Following the work of others [4, 11], we suggest that

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 241–256, 2003.

the coordination model of a multi-agent system should be tailored to application requirements. We consider that an application may have a set of very different goals, i.e. provide a variety of functionality. Also, we consider that this functionality may be best achieved at any one time by agents under the control of the application's designers or other accessible agents or processes within the open system.

We have developed a methodology in which we model cooperation to achieve an application's goals as the result of interactions between agents, without specifying which particular agents the interactions take place between. By raising *interactions* to the status of first-class design objects, we allow many agents, including those not existing at design time, to take on the role of participants in the interactions. The aim is that the most suitable agents within the open system can achieve the goals to the highest quality at any point in time.

With applications that have a set of differing goals, the agents that coordinate over those goals may appropriately use different coordination mechanisms. Choosing a single coordination model, such as a marketplace or a hierarchy, would not necessarily best fit the requirements. We therefore propose analysing each goal to determine the most suitable coordination mechanisms for use by agents cooperating to achieve the goal. We can then design agents that ensure application goals are achieved within the open system, based on the analysis results. Designing the multi-agent system for agent interactions where the participating agents are not known in advance places further demands on the analysis process.

In this paper, we demonstrate that our *assurance analysis* design process meets these demands, with a small example. In order, we present the following.

- A case study is provided to illustrate the methods described in this paper (Section 2).
- The case study requirements are analysed to derive the application *goals* and *preferences* as to how to best achieve the goals (Section 3.1).
- We discuss how a designer can choose from known agent coordination mechanisms to make sure the application is opportunistic and its design is justified (Section 3.2).
- Details of this approach are explained and illustrated by analysing the goals of the case study application (Section 3.3).
- After goals are matched to the coordination mechanisms which would best allow them to be achieved, we discuss how the designer can use this information to design the application agents. These agents will use the resources of the open system to achieve the application goals (Section 3.4).
- To show that the design is justified as we have claimed, we show how the design decisions made in the case study can be traced back to the requirements (Section 4).

Agent Interaction Analysis Our methodology, as a whole, is called *agent interaction analysis*. Using the methodology, a designer will perform broadly the following steps to create a multi-agent application functioning in an open system.

- Analyse the application requirements to discover the *goals* of the application and the *preferences* (or non-functional goals) that specify and/or quantify the priorities in achieving each goal (such as speed, accuracy of results etc.) Other information, such as how the goals are triggered in the system, e.g. by request from the user, and how goals can be decomposed into subgoals are also determined.
- The ideal target application design is then viewed in terms of agent interactions. The application can be seen as the achievement of the application goals within the open system due to cooperation between agents, each goal achieved being represented by an agent interaction. The agents taking part in the application are not specified as they cannot all be known about at design time ([8] also discusses designing systems that allow interactions between dynamically created agents).
- To achieve this target design, the designer adds agents to the open system to ensure the application goals are met.
- The designs of individual agents are tailored to match the preferences of the application, so as to best achieve the goals. The designer chooses between different models for each agent mechanism, where the models are described by third-party design patterns. The approach aims to preserve flexibility in the system as far as possible to allow full use of other agents in the open system that may more closely match preferences in their activity than the ones that the designer has added.

In this paper, we concentrate only on choosing the coordination mechanisms for the agents added to the system, which is a later stage of our methodology. As stated above, the coordination mechanisms most suitable for achieving a goal are those that tailor the interacting agents to best match the goal's preferences. See [10] for more on our methodology, and [13] for an overview of agent-oriented software engineering.

2 Case Study

To illustrate the design of multi-agent applications, we provide a case study that will be used throughout this paper. We assume that the case for using an agent-based approach for this application has already been made. In this section, we present the requirements document for an example application.

We require a collaborative weather mapping application. Using the application, a global weather map giving the current state is accessed and edited by various collaborating organisations. Contributors can add data they have gathered locally to the map in authorised locations. For example, one organisation may be authorised to add data concerned with a small local area, while another can add data for any location within a country. Authorisation is enforced to prevent accidental changes. Contributors and other organisations can access the weather data and be provided with predictions of weather at specified locations in the future. As with the contribution of data, different organisations have differing access rights to predictions on different locations.

Several services offer predictions based on the data, and the number of *predictors* available at any one time may vary, partly due to the load they each have on them. The predictor services vary in speed and in accuracy. They are represented as autonomous agents that accept prediction goals via published protocols. They must have access to the weather data to make the prediction, either by moving onto the system on which relevant data is stored or by repeated requests to the relevant sources. Prediction requests specify location and time (in the future), and the results should aim to be as accurate as possible. The application should be *opportunistic* in using open system resources, to ensure that operations are performed quickly or accurately, but should prioritise speed over opportunism in general. Authorisation for editing data and viewing predictions is governed by stored *access rights*, which some users are authorised to edit.

Of course, requirements are unlikely to be provided completely in a single document, and further requirements capture may require further processes. Appropriate requirements capture and analysis is beyond the scope of this paper, and so we summarise key points that have been identified by the designer in our case study.

- Users need to perform three operations using the application as a whole: contribute some weather data, view predictions based on this data and set access rights for other users.
- When contributing data to the map, a user wishes the data to be added as quickly as possible, access rights to be observed and the data to be reliably integrated into the map, especially as new data may be generated frequently.
- When viewing a prediction based on existing data, a user particularly wishes for the quality of the prediction (the probability of its accuracy) to be high, as well as wishing it to be quickly presented to them. Again, the access rights should be observed, but also the user of the application wishes to ensure that it has enough flexibility to use higher quality prediction services when they become available.
- In setting access rights, the user primarily wishes the operation to be performed quickly and in observation of access rights.

3 Designing Coordination for an Open Application

In this section we examine how agents can be designed to fit application requirements. In particular, we examine which coordination mechanisms would best match the preferences of application goals, to allow an agent added to the system, or, indeed, already existing in the open system, to best fulfil the application goals opportunistically. The justification for each design decision is made explicit in sections marked ‘Justification of Design Decision’.

Table 1. Goals identified by requirements analysis

Goal Name	Goal State Description	Preferences
Contributed Weather Data	A user has contributed weather data to the current map.	Security, Reliability, Speed
Viewed Prediction	A user has requested and received a prediction.	Security, Flexibiliy, Quality, Speed
Set Access Rights	A user has set the authorisation of another user for editing or viewing.	Security, Speed

3.1 Requirements to Design

Requirements engineering is the process of understanding and refining the requirements of an application. This may be aided by taking prospective users through usage scenarios [7], for example. Agent-oriented approaches to development should not force any change in an application's requirements. Therefore, the techniques used in requirements engineering for agent-based systems will be the same as those elsewhere [3, 12] (see Tropos [1] for an agent-oriented software engineering methodology based on requirements engineering).

A common requirements engineering technique is to describe the requirements in terms of functional *goals* it is intended to achieve and priorities and restrictions which, jointly, we call *preferences*. For brevity we exclude the requirements analysis of our case study and simply present the results in Tables 1 and 2 for goals and preferences respectively.

In Table 1, each goal identified is given a name, an end state and a list of associated preferences. The end state describes the state of the system in which the goal has been achieved. The associated preferences state those priorities and restrictions that should be observed while attempting to achieve the goal. A goal is, in fact, a class of *goal instances*; there will be many times in which a user will contribute weather data, for example, and each of those may be the contribution of data at a different map location. Each contribution is an instance of the *Contributed Weather Data* goal. In Table 2, each preference identified is given a name and a description.

Justification of Design Decision The goals and preferences are decided upon by being directly identified from the requirements.

For a complex application, goals can be divided into subgoals. A subgoal will inherit the applicable preferences from its parent goal. This is not discussed further here but see [10] for more.

Table 2. Preferences identified by requirements analysis

Preference Name	Preference Description
Security	Security of information prioritised.
Reliability	Reliability of achievement prioritised.
Flexibility	Flexibility for opportunistic behaviour prioritised.
Quality	Quality of results prioritised.
Speed	Speed of achievement prioritised.

3.2 Coordination Mechanisms

An application that is opportunistic will use the services available in an open system to best achieve the application goals. The designer of an open system application will implement this by adding functionality to the system that *coordinates* the services to best effect. In an agent-oriented approach, the functionality added is viewed in the form of agents. Agents coordinate activity between themselves and other services by using *coordination mechanisms*. For example, here are brief descriptions of two such mechanisms.

Trust An agent using this mechanism, based broadly on the one suggested by Marsh [9], keeps a record of the success of cooperating agents in achieving goals. Future choices of cooperating agents are decided by whichever agents have been most successful in the past. This allows the agent to choose between agents based on their *trustworthiness* [5]. For Contributed Weather Data and Set Access Rights goals, the successfulness of an attempt could be the speed at which the goal is achieved. For Viewed Prediction, the success is judged by the prediction's accuracy, which could be checked when the actual data for the predicted time and location is contributed. The mechanism may take longer to use than others as the agent will have to discover which potential cooperating agents are available before choosing one. There is a wide range of research on modelling trust [5], and we do not claim that this mechanism as described is better or worse than others.

Forced Cooperation An alternative method for an agent to find suitable cooperating agents is for the designer to implement agents with references to pre-selected agents that are known to be tailored to achieve a goal and are forced to accept requests for cooperation over that goal. This mechanism, similar to standard method invocation in object-oriented systems, prioritises speed and reliability over opportunism.

Different goals have different measures of success, as given by the preferences such as those given in the preceding section. In order for a coordination mechanism to be suitable for matching a preference, a mechanism must have two properties. First, it must be useful in choosing agents or other services in the open system whose activity best matches the preference. Second, it must match the preference itself. For example, in our case study, contributing weather data should occur as quickly as possible. A mechanism coordinating this goal should choose agents offering to edit the appropriate part of the weather map and should not slow the process down unduly by the act of coordination.

In *agent interaction analysis*, the designer performs the following steps to decide on which coordination mechanisms is most appropriate for each application goal.

1. The designer is supplied with a set of coordination mechanism definitions in a standard pattern language. This follows the approach of *design patterns* [6], in which abstract parts of a design are made available to designers to encourage *re-use* of well-founded designs. There are many agent coordination mechanisms suggested in the literature but we use only the above two as examples in this paper (for brevity).
2. For each application goal, the designer chooses a coordination mechanism most suitable for matching the goal's preferences. In order to aid comparison, the designer performs more detailed analysis of the mechanisms. This analysis is called *assurance analysis* and is described in the next section.
3. From the choices of coordination mechanisms for each goal, the designer decides on the coordination mechanisms for the agents that will be added to the open system. When these agents are added the appication will be instantiated. This step is called *collation* and is discussed in Section 3.4.

In the full methodology, the above steps are performed for mechanisms other than coordination mechanisms, e.g. plan execution mechanisms, action scheduling mechanisms etc. In order to aid the comparison described in the first step, the coordination mechanisms are written in a pattern language. Examples for trust and forced cooperation are shown in Tables 3 and 4. The tables describe eight aspects of each mechanism that may be revelant to application preferences.

3.3 Assurance Analysis

The choice between mechanisms, even when design patterns are provided, may be difficult because different stages of the mechanisms' use can have very different effects on an application's performance. For example, with a trust mechanism such as the one suggested above, the process of building up a large table of trustwothiness information on agents in the open system may be costly in terms of time and storage space. It would, however, be very useful in making coordination decisions on which agents to best cooperate with. In order to supplement the information given in the supplied patterns, an application designer can further analyse how well mechanisms meet the application requirements by breaking their operation into smaller parts.

Assurance analysis is a procedure in which to analyse how well a coordination mechanism matches an application goal's preferences, which therefore aids comparison of coordination mechanisms for the application. It takes the viewpoint of a *single* agent attempting coordination with unknown others, so as to ensure flexibility in making full use of the open system. In this paper, we use simple tables to provide the results of the analysis for brevity.

Table 3. A pattern for a coordination mechanism.

Part Name	Coordination
Model Name	Trust
Description	The agent using this mechanism checks the quality of any solution provided by an agent and uses these assessments to decide which offers to accept in the future. The checks can be either by observation of the state achieved, if the goal attempts to achieve a particular observable state, or by an independent production of the same information, if the goal attempts to derive some information.
Algorithms	1. Send requests to agents; 2. Wait for a suitable duration to receive offers; 3. If no offers received, resend requests; 4. If some (one or more) offers are received, assess them to determine which comes from the most trustworthy agent; 5. Accept the offer from the most trustworthy agent.
Priorities	A trust-based mechanism prioritises quality of solution and reliability in obtaining a solution.
Resource Use	The agent using the mechanism will need to possess quantitative assessments of the trustworthiness of other agents, which rise and fall depending on observed quality of solution. The agent will make observations or request extra information for each goal.
Support Required	As the agent using the trust mechanism algorithm must wait a specified duration, it requires a scheduling mechanism capable of this.
Scaling	With a large number of possible collaborators for a goal, the number of models possessed by the agent will also be large. The observational checks will add to the time taken to process each goal by the agent.
Applicability	Where the quality of the goal is able to be checked in some way and is of more importance than speed or the low use of resources.
Problems	A trust-based mechanism may add a significant amount of processing to each goal the agent seeks cooperators for.

In Tables 6, 7 and 8, we analyse and compare how well each of the two coordination mechanisms matches the preferences of each goal of our case study. The analysis attempts to be detailed to guide the designer in considering all aspects of the decision. Each cell in the tables is completed by the designer, and indicates whether a coordination mechanism is suitable for the goal being analysed, in some respect. Clearly two different designers may give different analyses, and so different entries into the tables, but the analysis still gives a *justification* for design decisions. Any more objective measure would be difficult as the priorities (*preferences*) of applications will vary widely.

Each of the tables is divided into four stages of execution. The top stage analyses the obtaining of information required to coordinate. The next stage down examines the updating of stored information on potential cooperating agents.

Table 4. A pattern for a coordination mechanism

Part Name	Coordination
Model Name	Forced Cooperation
Description	In this model, certain agents are required to cooperate over a goal on demand and are known to the agent employing this mechanism. The mechanism is approximately the same as message passing in (concurrent) objects.
Algorithms	To request cooperation from a forced agent, there is only one step. 1. Demand cooperation over the goal
Priorities	The model prioritises speed, reliability that the cooperators will be capable (through explicit design) and security in knowing the cooperator is pre-determined to be trustworthy.
Resource Use	Local information regarding the forced cooperation in agents is required by the agent employing this mechanism.
Support Required	The model requires mandatory adoption of goals in some agents providing the capability for this goal.
Scaling	Scales easily as it requires no communication beyond the minimum demand for cooperation, but does require suitable functionality to continue to be available within the application over time.
Applicability	This model is most applicable where security or reliability are of much greater importance than opportunism and where it is known that the forced functionality will always be available within the application.
Problems	Forced cooperation allows no flexibility in choosing cooperators so provides for no opportunism in exploiting the open system.

The third stage concerns the analysis made by the agent from the cooperating agent information. The lowest stage examines acting on the analyses made.

In each table, a column represents analysis of a single mechanism while a row represents analysis of a single preference. For each cell in the table, the designer asks the following question: "Does coordination mechanism X allow preference P to be matched in stage S." In each cell, the designer answers 'yes' or 'no' to the question. In the lowest stage, the designer can see the overall suitability of the mechanisms. In general, the mechanism that answers 'yes' to all the questions is the best to choose though a compromise may be necessary if no mechanism matches every preference.

For example, in Table 6, the Trust mechanism has a 'no' entry in the second stage for the preference to prioritise speed. This is because, at the stage of building up information on potential cooperating agents, the Trust mechanism could be slow and prioritises quality of solution over speed. It can be seen that the analysis of a mechanism with regard to a preference is independent of any goal, so the same analysis of the Speed preference is made in Tables 6 and 8, for instance. In Table 7, the Forced Cooperation mechanism is stated not to allow the prioritisation of flexibility or quality of solution from the point at which it gathers information for cooperation (the first stage). This is because it does not attempt to discover any more suitable agents in the system but instead relies on pre-selected agents.

The analyses made suggest that the Forced Cooperation mechanism is most useful for the Contributed Weather Data and Set Access Rights goals and Trust is best for coordinating over Viewed Prediction.

Justification of Design Decision The analysis of how well each coordination mechanism matches the preferences of each goal justifies the choice of mechanisms.

3.4 Collation

Once designers have decided which of the coordination and other mechanisms available best match the application preferences, they can use this information to decide on the design of agents. The agents, when added to the open system, should instantiate the application goals while matching the preferences.

In our example, we have decided that a Forced Cooperation coordination mechanism would be best for coordinating Contributed Weather Data and Set Access goals and Trust would be most suitable for coordinating Viewed Prediction goal. As there are no other mechanisms specified in this brief example, the most obvious agents to introduce are to have one tailored to adopting and coordinating over the Contributed Weather Data and Set Access goals and one for the Viewed Prediction goal.

We give the final agent design in Table 5, which includes both the coordination mechanisms and the goals that each of the agents will make offers to achieve if requested (the goals they *will adopt*). The first agent can only adopt the Contributed Weather Data and Set Access Rights goals because it requires pre-selected cooperating agents and will only have them for those goals. We allow the second agent to adopt any of the goals, as there is no reason to restrict it and therefore, for maximum opportunism of the application, we do not restrict it (though it will be restricted in implementation). The full criteria for deciding on the complete set of agents being added to an open system depends on the application-wide preferences given in the requirements. Techniques for assessing whether applications meet application-wide preferences is discussed in [2].

The above application of *assurance analysis* is fairly simple as our case study is also simple and contrived for the purpose of this paper. For this reason we cannot illustrate the full potential of the analysis here. For example, both of the Forced Cooperation and Trust mechanisms concentrate on choosing between suitable agents which leads to preferences being disallowed at the early stages (higher tables in the analysis) or not at all.

Justification of Design Decision The design of agents produced by collation is wholly based on the mechanisms determined to be best for each goal.

Table 5. Agents produced by collation

Agent	Coordination	Will Adopt
1	Forced Cooperation	Contributed Weather Data, Set Access Rights
2	Trust	All goals

Table 6. Assurance Analysis of Contributed Weather Data goal

Information Acquisition

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	yes	yes

Updating Models

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	**no**	yes

Information Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	**no**	yes

Acting on Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Reliability	yes	yes
Speed	**no**	yes

4 Judging the Design

When creating a multi-agent application, the design is justified by the requirements if the functionality is divided between agents in a way justified by the requirements. The dynamic nature of open systems means that any test performed on an implemented (opportunistic) open system application is unrepeatable and, therefore, of limited value [14]. Also, the application may make use of any functionality in the open system but, clearly, the only part of the application that can be tailored to the requirements is that which is known about at design time and whose form is under the control of the designer.

4.1 Tracing Backwards

The first agent in Table 5 has the functionality identified as most suitable for originators of two goals: *Contributed Weather Data* and *Set Access Rights*. The

Table 7. Assurance Analysis of Viewed Prediction goal

Information Acquisition

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Updating Models

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Information Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

Acting on Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Flexibility	yes	**no**
Quality	yes	**no**
Speed	**no**	yes

designer has chosen to merge the agents designed to coordinate over these goals, so that the organisation contains one agent (in the application set) tailored to providing this functionality rather than two. By simply identifying agents in the original requirements, the designer could, for example, have chosen to implement two agents that separately dealt with the two goals. To see why the decision to choose this particular organisation is justified, we can reason (trace) backwards from the *collation* stage, at which the final organisation is chosen.

1. At the start of the collation stage, the designer has decided that an agent tailored to coordinating over the Contributed Weather Data goal would have an architecture which used forced cooperation to coordinate. The designer has also decided that an agent tailored to coordinating over the Set Access Rights goal would have the same architecture. These are decisions on the most suitable architectures for agents initiating cooperation over the goals. *The decisions on the choice of most suitable architecture for each agent is the results of analysis, such as the* assurance analysis *described earlier, into how the agents can be tailored to match the requirements, and should, therefore, inform the choice of organisation.*

Table 8. Assurance Analysis of Set Access Rights goal

Information Acquisition

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Updating Models

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Information Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

Acting on Analysis

Preference	Trust	Forced Cooperation
Security	yes	yes
Speed	**no**	yes

2. In collation, the designer decides which agents make up the final organisation based on the coordination mechanism design decisions for the agents. It would be likely that at least one agent with the Forced Cooperation mechanism for coordinating over the Contributed Weather Data goal is implemented so that when an agent, or the user, wishes to achieve an instance of the Contributed Weather Data goal, there will be an agent tailored to doing so in the application. The same is true for the Set Access Rights goal. However having two agents with the same architecture is not necessarily the best organisational division, as having two agents will use up more resources. *Separation of agents into more than one agent type (architecture) in the organisation is not justified by the requirements if the architectures of the agents are similar enough, as long as the architectures are themselves justified by the requirements.*
3. To see that the architectures of the agents are justified we examine the *assurance analysis*. The assurance analysis for the Contributed Weather Data goal gives reasons why the most suitable coordination mechanisms for the goal is Forced Cooperation, based on comparing preferences of that goal with the operations of each mechanism. The Set Access Rights analysis comes to the same conclusion for that goal. *The choice of architectures for the agents was decided on the basis of the goal and application preferences.*
4. The Contributed Weather Data and Set Access Rights goals and their preferences were extracted from the requirements in the requirements analysis stage by examining the scenarios, entities and goals mentioned in the requirements.

4.2 Opportunism

The other aspect of justification for designs is that it is restricted as little as possible in its opportunism. The two agents in Table 5 differ greatly in their interoperability. The first agent is heavily restricted in its operations while the second is highly interoperable. We claim that the opportunism of each is restricted only as far as the requirements demand and the design decisions are, therefore, justified. We examine the design decisions restricting or allowing opportunism for each agent below.

1. The first agent is heavily restricted in its activity.
   (a) The preference to validate authorisation to edit the weather map and access rights and the emphasis on reliability in contributing to the map require that the agent uses only trusted agents in making alterations. This can either be achieved by checking the agent's actions to determine their reliability or by always using standard cooperating agents chosen by the designer. Speed is more important than interoperability so the designer chose the latter option (the Forced Cooperation coordination mechanism).
   (b) Using Forced Cooperation requires that references to cooperating agents be known in advance. Thus, only the goals for which references are provided can be attempted by the agent, which are Contributed Weather Data and Set Access Rights in this case. The agent will only offer to adopt these two goals.
   (c) These restrictions are derived from the requirements (goals and preferences).
2. The second agent has very little restriction on its behaviour.
   (a) The preferences on getting accurate predictions in prediction viewing operations lead the designer to use interoperation to find the most suitable cooperating agents at each instance. To decide between cooperating agents, the agent uses a trust-based mechanism giving information on the previous likely accuracy of results.
   (b) There is no reason to prevent the agent from being given the ability to coordinate over other goals, and so it is given the ability to do so.
   (c) The requirements encourage opportunism in this case, and the design of the agent reflects this.

5 Conclusions

In this paper, we have shown how a designer can choose coordination mechanisms for agents implementing an application in an open system. We have shown that these decisions can be analysed to determine that the design does meet the requirements, and allows for flexible use of the functionality available in the open system at any one time.

Our methodology was developed to allow designers to produce opportunistic open systems applications whose design is fully justified by the application requirements. We do not believe this is currently achieved by other agent-oriented software engineering methodologies.

Future work will examine how the infrastructure supporting agents can also be designed to cohere with the guiding concepts identified above. One minor problem to be addressed with the methodology, however, as with any that attempts to analyse all points at which an application could take advantage of available functionality, is keeping the volume of the specification to a manageable level. More detailed results of the case study, and further details of the methodology, are available from the first author.

Acknowledgements

This work was carried out while supported by the myGrid project (EPSRC reference GR/R67743/01) at University of Southampton.

References

[1] J. Castro, M. Kolp, and J. Mylopoulos. A requirements-driven development methodology. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE-01)*, Interlaken, Switzerland, 2001. 245

[2] P. Davidsson and S. Johansson. Evaluating multi-agent system architectures: A case study concerning dynamic resource allocation. In *this volume*, Madrid, Spain, 2002. 250

[3] A. M. Davis. *Software Requirements: Objects, States and Functions*. Prentice Hall, 1993. 245

[4] V. Dignum, H. Weigand, and L. Xu. Agent Societies: Toward Frameworks-Based Design. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proceedings of the Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, pages 25–32, Montreal, Canada, 2001. 241

[5] R. Falcone and B. S. Firozabadi. The challenge of trust: The Autonomous Agents '98 Workshop on Deception, Fraud and Trust in Agent Societies. *Knowledge Engineering Review*, 14(1):81–89, 1999. 246

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse of object oriented design. In *Proceedings of ECOOP'93, Kaiserslautern, Germany*, 1993. 247

[7] P. Haumer, K. Pohl, and K. Weidenhaupt. Requirements elicitation and validation with real world scenes. *IEEE Transactions on Software Engineering*, 24(12), December 1998. 245

[8] M. N. Huhns. Interaction-oriented programming. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000. 243

[9] S. P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1994. 246

[10] S. Miles, M. Joy, and M. Luck. Designing agent-oriented systems by analysing agent interactions. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000. 243, 245

[11] A. Omicini. SODA: Societies and infastructures in the analysis and design of agent-based systems. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000. 241

[12] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Proceeding of the Twenty-Second International Conference on Software Engineering (ICSE-00)*, 2000. 245

[13] M. Wooldridge and P. Ciancarini. Agent-oriented software engineering: The state of the art. In P. Ciancarini and M. J. Wooldridge, editors, *Proceedings of Agent-Oriented Software Engineering 2000 (AOSE 2000)*, 2000. 243

[14] F. Zambonelli and H. V. D. Parunak. Signs of a revolution in computer science and software engineering. In *this volume*, Madrid, Spain, 2002. 251

Specification by Refinement and Agreement: Designing Agent Interaction Using Landmarks and Contracts

Hans Weigand[1], Virginia Dignum[2, 3], John-Jules Meyer[3], and Frank Dignum[3]

[1] Infolab, Tilburg University
PO Box 90153, 5000 LE Tilburg, The Netherlands
weigand@kub.nl
[2] Achmea
P.O. Box 866, 3700 AW Zeist, The Netherlands
virginia.dignum@achmea.nl
[3] University Utrecht
PO Box 80089, 3508 TB Utrecht, The Netherlands
{virginia,jj,dignum}@cs.uu.nl

Abstract. In this paper, we argue that multi-agent systems developed to model and support organizations must on the one hand be able to describe and realize the goals and structure of that particular organization and on the other hand allow for autonomous behavior of participating actors. The agent autonomy should include collaboration autonomy, which means that agents can decide among each other how they want to cooperate. We present a model for agent societies and two techniques that can be used to achieve the described objectives: landmarks and contracts. The effects of these techniques on different agent society types are explored as well.

1 Introduction

Agent Societies emerge from the idea that agent interactions occur not just by accident but aim at achieving some desired global goals. That is, there are goals external to each individual agents that must be reached by the interaction of those agents. Desired behavior of a society is therefore external to the agents. Furthermore, we assume that in such a setting, social structure is determined by organizational design and not dependent on the agents themselves. We use the term Organizational Model (OM) to describe this global behavior and structure.

However, the behavior of individual agents is motivated from their own goals and capabilities, that is, agents bring in their own ways into the society as well. The actual behavior of the society emerges from the goal-pursuing behavior of the individual agents within the constraints set by the Organizational Model. From the society point of view this creates a need to check conformance of the actual behavior to the desired behavior and this has several consequences. Firstly, we have to make explicit the

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 257-269, 2003.

commitments between agents and the society. An actor is an agent performing one or more roles in the society. The objectives and capabilities of the role are described in the OM and the actual agreements of the actor concerning its interpretation of the role are explicitly described in a social contract. We use the term Social Model to refer to the social contracts that hold at a given moment in a society. Secondly, actual interaction between actors must also be made explicit, which can be done through (bilateral) contracts as well. We call this the Interaction Model (IM). Checking the conformance of actual behavior to the desired behavior can now be realized in two steps:

- Checking whether the contract set up by two or more agents conforms to the OM. The OM can be more or less elaborate, depending on the type of society. Typically, it does not provide more than a few ìlandmarksî that describe the main features (conditions, obligations, and possibly partial plans) of interaction between roles.
- Monitoring whether the actual messaging behavior conforms to the contract. This is primarily a responsibility of the agents themselves. Depending on the type of society, the OM can constrain the monitoring or even assign it to the society.

In an Agent Society modeled from an organizational perspective, both the specific architecture of the individual agents and their motives to perform a role are ignored. In principle, an agent societies model is possible to be populated by any sort of agent (that is, first you have agents and then they decide to join a society), but the role description must be rich enough to allow for the design of agents specific to enact that role (that is, first you have a society and then you design agents to fulfil its roles).

The structure of this paper is as follows. In section 2, we describe the Agent Society model that we start from. Section 3 discusses the requirements on the specification of the interaction model that result from the society characteristics. Sections 4 and 5 present respectively, an Organizational Model and a contract concept that can meet the requirements. In section 6, the effects of these techniques on the different agent society types (markets, networks, hierarchies) are explored. Section 7 is the conclusion in which we evaluate the results and indicate some topics for future research.

2 Agent Societies

The aim of agent societies is to describe multi-agent systems from an organizational (collective) perspective. Our work combines a society model similar to [2] and [12] with contracts, as a formalism to specify cooperation between agents and society, as proposed by [17], [6] and [1]. In [11] we introduce a model for agent societies, which takes an organizational perspective and describes a society in three levels:

- **Organizational model (OM):** describes the desired or intended behavior and overall structure of the society from the perspective of the organization in terms of roles, interaction scripts and social norms:

- **Social model (SM):** populates the organizational model with specific agents mapped to roles through a social contract. Social contracts describe the agreed behavior for an agent within the society in terms of externally observable events. Agents enacting a role are called actors.
- **Interaction model (IM):** specifies interaction agreements between actors. Describes the actual behavior or the society.

Figure 1 depicts the different models. The model provides a separation between the intended, ideal behavior of the society as sought by the designer (organizational model) and the actual emergent behavior resulting from the autonomous action of participating agents (interaction model). We focus on designed societies (that is, societies that are explicitly developed for a certain aim, in opposition to societies that emerge from the local co-existence of a group of agents). In these societies the organizational aims are usually identified a priori and the society model must allow for predictability and verification. The organizational model describes the intended interactions between agents that from the perspective of the society designer will lead to the expected behavior of the society as a whole. However, actual interactions depend on the behavior and intentions of the actual agents involved at a given moment of the society life.

We assume that the designer of the society will usually have no control over the design of the participating agents (open societies). Therefore, verification of society goals cannot be based on assumptions about the internal construction of agents. The social model describes explicitly the agreements between an individual agent and the society concerning the expected/allowed activity of the agent from the point of view of the society. These ì labor contractsî will specify obligations and rights that an agent is entitled to as enactor of a society role (that was described in the organizational level). In principle, agents are free to pursue other of its own goals as far as those are not in violation of its social contract. In [5] we consider the implementation of agent societies in a given agent language (3APL, [13]), that is, how can the architecture of individual agents incorporate social concepts described in an organizational model of agent societies.

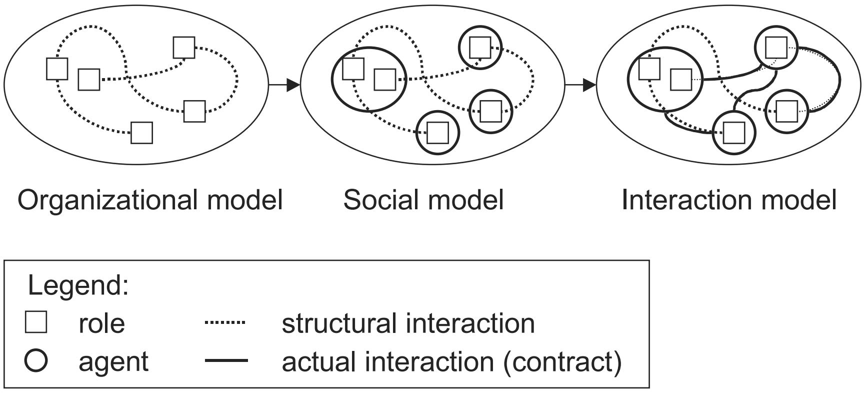


Fig. 1. Organizational framework for agent societies

Actual interactions between agents are specified in the interaction model. These interactions can follow fixed protocols (e.g. the contract net protocol) but can be the result of communication between agents in which the interaction forms, results and duties for all parties have been negotiated. The type of interaction and the freedom for agents to negotiate specific interaction forms depends of the society model. In network models it is possible to create new interactions but market and hierarchical societies provide at least some fixed forms.

In order to develop a common understanding and a feeling of the possible applications of the agent society framework, it is necessary to develop scenarios in different areas, such as e-commerce, co-operative work and mediation. The agent society framework is especially suitable in situations where control and initiative are distributed and social order is an issue. In human organizations and societies, norms and conventions are used to specify the behavior that society members are expected to conform to. Such norms are usually backed by a variety of social institutions that enforce law and order, monitor for and respond to emergencies and prevent and recover from unanticipated disasters. In this way, citizens of civilized societies can utilize relatively simple and efficient rules of behavior, offloading the prevention and recovery of many problem types to social institutions that can handle them efficiently and effectively by virtue of their economies of scale and widely accepted legitimacy. The virtue of an institution resides in its potential to lend legitimacy to its members and security to its traders by establishing norms. Several researchers have recognized that the design of agent societies can benefit from abstractions analogous to those employed by our robust and relatively successful societies and organizations [4, 7, 12]. Electronic institutions can engender trust through certification of an agent and by the guarantees that it provides to back collaboration. Furthermore, the electronic institution can also function as the independent place in which al types of agent independent information about the interaction between the agents within the society is stored [8].

3 Requirements for Interaction Specification

In order to represent interactions between agents in such a context, a framework is needed that is able to specify sequences of interaction scenes

- independently from the internal design of the agent (internal autonomy requirement) and
- without fixing the interaction structures completely in advance (collaboration autonomy requirement).

Contracts are an adequate means for the explicit specification of interactions [17]. Contracts describe externally observable events and the contract specification language is independent from the internal agent specification.

As in human societies, existing norms and conventions do not cover all possible interactions between agents. That is, it is not possible to completely specify all interactions between agents a priori because we cannot account for environment and agent change during the life of the society. Therefore, there is a need to allow for the ërun-timeí specification of new interaction scenes. If contracts are used as a specification mechanism, this implies that the society must be able to provide partial specifications of contracts. In principle, this can be done in several ways. For example, Kumar and Cohen describe interaction scenes using landmarks, which are sets of propositions holding in the state [14]. The landmarks are specified beforehand, but it is left to the agents how to achieve these landmarks. Some interaction scenes may allow for the negotiation of more specific interaction scenes, which involves the definition of new interaction scenes and landmarks (specialization of a conversation protocol in terms of [14]).

Fundamentally, a tension exists between the goals of the society designer and the autonomy of the agents. When the society designer can specify the agent interactions in more detail, it is possible to check and guarantee more of the requirements at design time. Formal analysis of the interaction structures is possible, for example, to check basic audit principles as in [3]. It may be necessary to ensure the legal validity of the interactions that certain rules are followed. It can also be important for new coming agents that want to join the society that they know the norms and the interaction protocols used.

On the other hand, there are good reasons to leave the agents with some degree of freedom. We call this the collaboration autonomy requirement, which basically says that the agents who want to collaborate are free in choosing their own way of achieving this collaboration. Advantages of collaboration autonomy are:

- **Extensibility**: not everything needs to be, and often cannot be known beforehand. The society can evolve smoothly when new interactions or ways to perform these interactions are being developed.
- **Flexibility**: specifying a certain interaction structure beforehand often means that design choices are made that are not really necessary. Forestalling these choices gives the agents the possibility to make these choices themselves, geared to their particular situation.
- **Perusability**: not only is it possible to develop new interaction structures within the society, but it is also easier to reuse interaction structures across societies.

It must be clear that the collaboration autonomy (as any autonomy requirement) is always relative. Certain rules or structures must be present, so that the agents can build on top of that. For example, in a market society it must be possible to arrive at a transaction. It is often not important how the transaction is achieved, but it must be clear at some point that a business transaction has been closed. Sometimes it is important how the transaction is achieved. If the number of suppliers and/or buyers is very large and efficiency is critical, an auction mechanism is most effective, and this means that the interaction structure is highly fixed.

In the agent society model, two levels of specification for interactions are provided. The OM provides a script for interaction scenes according to the organizational aims and requirements and the IM, realized in the form of contracts, provides the interaction scenes such as agreed upon by the agents. It is the responsibility of the agents to check that the IM is a refinement of the OM (this can be supported by formal verification). It is also the responsibility of the agents to ensure that their actual behavior is in accordance with the contracts (the IM ñ they can use a monitoring agent or notary services provided by the society for that). However, it is the responsibility of the society to check that the agents fulfill these responsibilities (this is realized by means of social contracts, cf. section 5).

ìRefinementî is defined by the OMG as ìa more detailed description that conforms to another (its abstraction)î. It is proposed as a technique in Model Driven Architecture (MDA) so that more detailed descriptions are built in a systematic way from abstract ones. For example, platform independent models (PIM) which are refined into platform-specific models (PSM). For agent-oriented design methods at the level of agent societies, the interoperability problems are of course just as relevant as for the OMG. What we argue here, from an agent perspective, is that it is not sufficient to define a technical notion of refinement, but that it is also necessary to recognize different actors, to identify responsibilities of these actors, and what is needed at facilitation level to make it work.

4 Organizational Model

The organizational model specifies the structure of an agent society in terms of externally observed behavior and components. An organizational model is a tuple OM = (R, CF, I, N), where R is the set of role descriptions, CF is the communicative framework, I is the interaction structure and N is the set of society norms or rules. The elements of OM can be referred to by:

roles(OM) = R
comm-framework(OM) = CF
interactions(OM) = I
norms(OM) = N

Society goals are not part of the organization model but form the background to the society definition and are represented by the goals of the roles. That is, the goals of a society are specified in terms of roles that correspond to the different stakeholders in the domain. Moreover, the organizational model can be seen as split into two parts: facilitation and operation. The facilitation layer provides the backbone of the society and consists of institutional agent roles, which are designed to enforce the social behavior of agents in the society and assure the global activity of the society. The operational layer models the overall objectives and intended action of the society and consists of domain related roles. The operational layer is always domain and application dependent whereas the facilitation layer depends on the cooperation characteristics of the environment and can be reused across domains (this is one reason why the recognition of this layer is useful during a design process). Typically, facilitation issues are suitable for outsourcing to an external party ([8]). The organizational model does not need to make any formal difference between facilitating agents and operational agents (we can use the same models and logics), but usually, the facilitating agents will be more controlled by the institution.

In the following, we will not describe all the components (see [9] for a more elaborate treatment), but focus on the Interaction Model using the example of conference organization. The global goal of the conference society is to realize a conference. This goal can be split into a facilitation component that aims at the organization of the conference, and an operational part where the conference program is generated and consumed. *Operational* stakeholders in this society, specified as roles in the operational component of the model, are authors, PC members and participants. The goal of author is to get papers accepted, the PC member aims at assuring the quality of the program and the participant hopes for a high quality to consume. *Facilitation* activities can be described in terms of an organizer role that administrates the conference and a chairperson role responsible to regulate conference sessions.

4.1 Interaction Structures

The possible actions of a role determine interactions with other agents. These interactions are articulated into (institutionalized) interaction patterns between agents that establish the possible dialogues and interchanges between agents. Such patterns are designed in such a way that the society aims are achieved. Examples are the negotiation dialogue needed to close a business transaction, or the workflow supporting a delivery process. We define an **interaction structure** as a pair I = (S, T) where S is a set of scenes (represented by interaction scripts) and relation T: S×S gives the scene transformation matrix. For s, sí ∈ S, (s, sí) ∈T, means that sí can be reached from s.

The scene transformation induces a partial ordering T^* on the set of scenes. Scene transformation T has consequences for *roles(OM)* in the sense that actors will follow different paths according to their roles. Furthermore, the performance of an actor in a given scene will have consequences to the roles that actors will play in other states. For example, the interaction structure of the conference society is depicted in Fig. 2. Note that the graphical representation is very similar to (and in fact inspired by) the work of Sierra [12]. However, an important difference is that protocols are fixed in that work, while our framework is based on the principles of refinement and collaboration autonomy.

An **interaction script** describes a scenario of activity, that is, how roles interact and evolve in the context of a scene. Interaction scripts serve as a blueprint for the actual interactions between actors. Landmarks are logical expressions that describe the characteristics (for instance, goals and action plans) of the scene. The level of specification of landmarks determines the degree of freedom the actors have about their performance. Norms are deontic expressions that regulate the activity of actors.

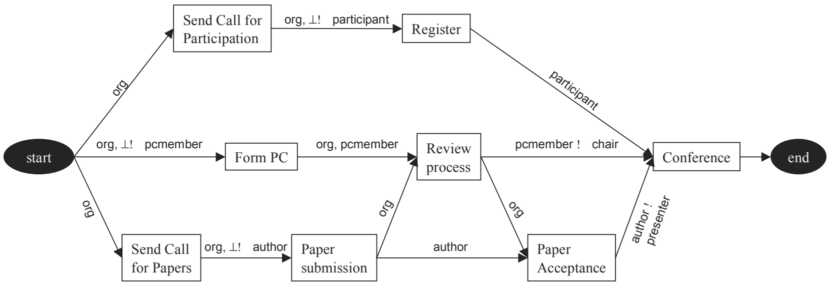


Fig. 2. Interaction structure for ëconferenceí society

```
Review process
Roles:
            Organizer    (1)
            PCmember   (N)
Landmarks:
            ∀ p ∈Papers,                   get_reviews(p, review1, review2).
            ∀ p ∈Papers,                   paper_decision(p, decision, review1, review2).

            DONE(Organizer, get_reviews(P, Rev1, Rev2)) ≡
                DONE(Organizer, assign_paper(P, PCmemberA, DeadlineR) BEFORE DeadlineA)
                AND DONE(Organizer, assign_paper(P, PCmemberB, DeadlineR), BEFORE DeadlineA)
                AND DeadlineA BEFORE DeadlineR
                AND DONE(PCmemberA, receive_review(Org, P, Rev1) BEFORE DeadlineR)
                AND DONE(PCmemberB, receive_review(Org, P, Rev2) BEFORE DeadlineR)
Norms:
            PERMITTED(Organizer, assign_paper(P, Reviewer, Review_deadline) ).
            OBLIGED(Reviewer, receive_review(Org, P, Rev) BEFORE Review_deadline).
            OBLIGED (Organizer, paper_decision(P, Decision, Rev1, Rev2) BEFORE Decision_deadline).
```

Fig. 3. Interaction scene ëReview processí

An interaction script, IS $\in$ S, is defined by IS = (P, CFí, L) where:

- P is the set of participating roles, P $\subseteq$ *roles*(OM),
- CFí is the communicative framework used in the scene, possibly a specialization of *comm-framework* (OM),
- L are the landmarks describing the interaction (propositions that hold for the scene, including the norms):

For example, in the conference society the interaction script for scene ëReview processí is partially described in figure 3. The arity (1 or N) of roles indicates the allowed number of different actors that can play the same role in the scene. Transformations indicates the possible consequences for actors of a role in future interactions (where $\perp$ indicates the disappearance of a role). As shown, landmarks can be more specific than just goals; in this example organizational design makes explicit how reviews are supposed to be obtained. Actors are in this case still free to decide about the assignment of papers (for example, letting reviewers indicate their preferences), about how to deal with missing reviews or about the acceptance or not of papers (for instance, do reviewers get to vote about papers or does the organizer decide alone). Such decisions are to be fixed as contracts in the scene play agreements in the Interaction Model.

5 Matching OM and Agent Autonomy: Contracts

The Organizational Model as described above is not enough to describe the activity of an agent society. The OM provides the specification of the society behavior as desired by the society designers. However, the actual behavior of the society emerges from the goal-pursuing behavior from the individual agents within the constraints set by the Organizational Model. From an organizational point of view this creates a need to check conformance of the actual behavior to the desired behavior. In the **Social Model** (SM) the social contracts that hold at a given moment in a society are described. These contracts make explicit the commitments between actors and the society, that is, specify all the aspects that govern and determine the interaction of a member with the rest of the society. Depending on the capabilities of the agents and on the organizational constraints, such contracts can be negotiated by the agents seeking admission. The social model contains the instantiation of roles to actual agents (this combination of agent and role is called an **actor**).

Furthermore, actual interactions between actors must also be made explicit. We call this the **Interaction Model** (IM). In the agent model interaction scenes are instantiated or created as a result of previous scenes for individual agents. Interaction scenes are described for generic agent roles and can be instantiated for enacting agents. A scene instantiation is described in a **contract** between the interacting agents. In the following, we collect the most important requirements for a contract language ñ a formal logic that meets these requirements is described in [10].

5.1 Contract Requirements

The interaction structure as specified in the OM, gives a partial ordering of interaction scenes and the consequences of transitions from one to the other scene to the roles (creation, destruction, etc). That is, the interaction structure indicates the possibilities of an agent at each stage and the consequences of its choices. Explicit representation of commitments is necessary in order to know how to proceed if the norm is violated but first of all to inform the agents about the behavior they can expect from the other agents. In this way, coordination can become possible. A formal language for contract specification is needed in order to be able to evaluate and verify agent interaction.

Contract rules usually do not express an immediate obligation, but the obligation to achieve a certain result before something else has happened. The fact that usually contract obligations refer to a deadline indicates that temporal aspects are of paramount importance for the specification of contracts. Furthermore, because agents are autonomous entities that can decide on the course of action they will take, different futures most be considered at any moment. Deontic logic provides possibilities to reason about violability of norms, that is, about how to proceed when norms are violated. In systems where agents are assumed to be autonomous and intelligent, norm violation must be considered (agents can deliberately choose to violate a norm if they think that their goals are better achieved in this way).

Table 1. Coordination in agent societies

	Market	**Network**	**Hierarchy**
Type of society	Open	Trust	Closed
Agent ëvaluesí	Self interest	Mutual interest/ Collaboration	Dependency
Facilitation roles	Matchmaking Banking	Gatekeeping Registry Matchmaking	Interface Control

Furthermore, verification of the behavior of an open society, where the design of participating agents cannot be controllable, must be based on the externally observable effects of agent actions. That is, from the society perspective, different actions that bring about the same state of affairs in the world cannot be distinguished.

6 Society Types

The way organizational coordination models achieve coordination is determinant to the motivation of coordination in agent societies. Table 2 derived from [11] gives an overview of the characteristics of agent societies with different coordination models.

In **markets**, agents are self-interested (determine and follow their own goals) and value their freedom of association and own judgement above security and trust issues. Openness is thus per definition a feature of markets. Facilitation is, in the most extreme case, limited to identification and matchmaking activities. Interaction in markets occurs through communication and negotiation.

Network organizations are built around general patterns of interaction or contracts. Relationships are dependent on clear communication patterns and social norms. Agents in a network society are still self-interested but are willing to trade some of their freedom to obtain secure relations and trust. Therefore, agents need to enter a social contract with the network society in which they commit themselves to act within and according to the norms and rules of the society. The society is responsible to make its rules and norms known to potential members. Coordination is achieved by mutual interest, possibly using trusted third parties, and according to well-defined rules and sanctions.

Finally, in a **hierarchy** interaction lines are well defined and the facilitation level assumes the function of global control of the society and coordination of interaction with the outside world. In a hierarchy, agents are cooperative, not motivated by self-interest and all contribute to a common global goal. Coordination is achieved through command and control lines. Agents are cooperative and not motivated by self-interest. Such agents are said to be benevolent, that is, agents are assumed to always attempt to do what is requested from them.

Let us consider now the way agent interaction is defined in each of these society types using the agent society model. In a **market**, the OM is relatively simple. It contains the roles of Buyer and Seller, as well as some facilitating roles such as Market Owner, Matchmaker and Bank. The Interaction Scenes fix the interaction based on standard protocols, such as auctioning protocols. In a typical market, the agents cannot refine these protocols, in other words, the contracts at IM level are identical to the scenes (of course, the parameters are different for each instance, but the contract structure is fixed). However, a market may also allow for refinement (in which case the society will start to resemble the network type). The Social Contract (SM level) specifies for an agent if it can play the role of Buyer, Seller, or both. The social contract also specifies what can happen in the case of misbehavior. However, the parties themselves are responsible for monitoring the actual performance. If the counterparty does not behave according to the scene/contract, and the issue is not solved bilaterally, one party can raise a claim with the Market Owner. Depending on the Social Contract, the Market Owner may then decide to sanction the non-behaving party.

In a typical **hierarchy**, the OM has a rich differentiation. It contains various functional role descriptions and the Interaction Scene binds these roles together into process models. One or more Controller roles must be defined that have the task of monitoring the performance of a functional role, comparing it with the organizational goals or some targets derived from them, and possibly initiating some remedial action. The extreme case is that the Controller role is performed by one agent and the OM does not allow for refinement. In such a situation, the advantages of an agent architecture such as local autonomy and flexibility become problematic. The other extreme is that the Controller role is distributed over many agents, and that refinement of contracts is possible. In this situation, the hierarchy starts to resemble a network. Normally, the structure of the hierarchy will be somewhere between these extremes, depending on the level of standardization that is desirable in the given environment. The assumption that agents are cooperative, that is, follow-up requests, is preferably not taken for granted, but put explicitly in the Social Contract. Agents are supposed to follow-up all requests authorized directly or indirectly by the Social Contract. On this point, the hierarchy is fundamentally different from both the market and the network.

The **network** type is the type that maximizes collaboration autonomy. The Organizational Model can be simple, including not more than a role Participant and some facilitating roles such as Notary and Monitor, or it can be more elaborate, such as the Conference Organization example described above. The function of the Notary is to verify the correctness of contracts (whether they conform to the scene descriptions), and to register them. The role of the Monitor is to validate the contract, that is, to monitor actual performance. This role can also be assigned to the parties themselves. If actual performance deviates from the contract, typically the contract will indicate remedial action. If there is breach of contract, the parties can report this to the Network Owner or to some Reputation role. This needs to be defined in the Social Contract to which each agent in the network has to commit.

It can be observed in this discussion is that both markets and hierarchies, however different in nature, can make use of the same two basic coordination mechanisms. These basic mechanisms are *standardization* and *collaboration autonomy*. Collaboration autonomy corresponds with what Mintzberg [15] has called *mutual adjustment*, and with respect to standardization he has distinguished several types of standardization (of work processes, outputs, skills, or norms). The sixth organizational coordination mechanism distinguished by Mintzberg is direct supervision, which we would like to generalize to *authorization* and which is the basic structure of hierarchies.

7 Conclusion

In this paper, we have described an approach to the specification of interactions that respects both the internal autonomy of agents as well as the collaboration autonomy. In order to profit from the flexibility of autonomous agents while at the same time ensuring the robustness of the complete system, we have argued for the need to define an organizational structure for the multi-agent system (similar to real-life organizations). This organizational structure is based on the goals of the overall system and indicates the behavioral boundaries for the individual agents within the system. We have described an Organizational Model that allows the designer of the Agent Society to specify some very basic structures (landmarks) to be respected by all agents within the society. These basic structures include role definitions as well as interaction scenarios.

Because agents come in with their own goals and capabilities, we need to specify how the agent fits into a certain role it takes in the multi-agent organization. To this end, a social component (the Social Model) has been added to the agent architecture. This component takes care that the goals of the agent fit within those for the role it plays and that the agent fulfills all norms belonging to the particular role. We have shown how the Social Model and the Organizational Model differ for the various society types (hierarchies, networks, markets). One conclusion that can be drawn from that discussion is that refinement is most typical in a network type of society, but it can be combined with the other types as well. The extreme market and hierarchy types do not support collaboration autonomy, but they can introduce it to some degree, thereby evolving into a network.

The combination of ìrefinementî and ìagreementî in our proposal is rather unique. Refinement in itself is not new; for example, it is also used under the name of ìlevelingî in AUML [16], and we already mentioned the use of refinement in the Model-Driven Architecture of the OMG. However, the view developed in this paper, refinement is more than an abstraction device: it is a matter of the proper balancing of responsibilities.

For the society, it is important that these contracts adhere to the landmarks given by the Organization Model. Therefore, the specification languages must be formalized syntactically and semantically so that formal verification is possible. We have developed a branching time deontic logic called LCR [10] that can provide the logical semantics of both the Organization Model and the contracts.

Using the models presented in this paper, and the LCR logic, it becomes possible to give a precise and implementable specification of agent societies. One important topic for future research is to demonstrate precisely that, that is, the development of architectures and design patterns for various types of agent societies.

References

[1] Andrade, L., Fiadeiro, L, Gouveia, J., Koutsoukos, G. Lopes, A. and Wermelinger, M.: Coordination Patterns for Component-Based Systems. Musicante M., Haeusler H. (Eds.): Proc. of Brazilian Symposium on Programming Languages, UFPR 2001, (2001), B: 29-39.

[2] Artikis, A., Pitt, J.: A Formal Model of Open Agent Societies. Proc. Autonomous Agents 2001, (2001) 192-193.

[3] Bons, R., Lee, R., Wagenaar, R., Wrigley, C.: Modeling Inter-organizational Trade Procedures Using Documentary Petri-nets. Proc. Hawaii International Conference (1995).

[4] Castelfranchi, C.: Engineering Social Order, Omicini, A., Tolksdorf, R., Zambonelli, F., (Eds.) Engineering Societies in the Agents World, First International Workshop, ESAW 2000, Berlin, Germany, LNAI 1972, Springer-Verlag (2000), 1 ñ 19.

[5] Dastani, M., Dignum, V., Dignum, F.: Organizations and Normative Agents. Submitted to the First Eurasian Conference on ICT, Teheran, Iran, October, (2002).

[6] Dellarocas, C.: Contractual Agent Societies: Negotiated shared context and social control in open multi-agent systems. Proc. WS on Norms and Institutions in Multi-Agent Systems, Autonomous Agents-2000, Barcelona (2000).

[7] Dignum, F.: Autonomous Agents with Norms. In AI and Law, (**7**), (1999) 69 ñ 79.

[8] Dignum, V, Dignum., F.: Modeling agent societies: co-ordination frameworks and institutions. In: P. Brazdil, A. Jorge (Eds.): Progress in Artificial Intelligence. LNAI 2258, Springer-Verlag, 2001.

[9] Dignum, V., Meyer, J-J., Weigand, H., Dignum, F.: An Organizational-oriented Model for Agent Societies. In: Proc. Int. Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), at AAMAS, Bologna, Italy, July, (2002).

[10] Dignum, V., Meyer, J.-J., Dignum, F., Weigand, H.: Formal Specification of Interaction in Agent Societies. 2nd Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS), Maryland, Oct, (2002).

[11] Dignum, V., Weigand, H., Xu L.: Agent Societies: Towards framework-based design. Wooldridge, M., Weiss, G., Ciancarini P. (Eds.): Agent-Oriented Software Engineering II, LNCS 2222, Springer-Verlag, (2002) 33-49.

[12] Esteva, M., Padget, J., Sierra, C.: Formalizing a language for Institutions and Norms. Proceedings of the 8th International Workshop on Agent Theories, Architectures and Languages, ATAL-2001, Seattle, (2001).

[13] Hindriks, K., De Boer, F., Van der Hoek, W., Meyer. J-J.: Agent Programming in 3APL. *Autonomous Agents and Multi-Agent Systems*,2(4):357ñ401, (1999).

[14] Kumar, S., Huber, M., Cohen, P., McGee, D.: Towards a Formalism for Conversation Protocols Using Joint Intention Theory. Chaib-draa, B., Dignum, F. (Eds.): Computational Intelligence Journal, Special Issue: Agent Communication Language, (2002). *To appear.*

[15] Mintzberg, H. Mintzberg on Management. The Free Press, New York (1989).

[16] Odell, J., Van Dyke Paranak, H, and Bauer,B., Extending UML for Agents. Proc. AOIS Workshop, Austin, July (2000), pp.3-17.

[17] Verharen, E.: A Language/Action Perspective on the Design of Cooperative Information Agents. Ph.D. Thesis, Tilburg University, (1997).

An Agent and Goal-Oriented Approach for Virtual Enterprise Modelling: A Case Study

Zhi Liu[1] and Lin Liu[2]

[1] Department of Computer, Zhejiang University of Technology
Hangzhou, P.R.China, 310014
lzhi@cs.toronto.edu
[2] Department of Computer Science, University of Toronto
Toronto, Ontario, Canada, M5S 3H5
liu@cs.toronto.edu

Abstract. Virtual enterprise established to satisfy the requirements of market is a one-off, dynamic distributed organization. In a virtual enterprise, partners who are selected form an alliance to fulfill goals that otherwise canít be achieved for the limitation of individualís capabilities. Modelling virtual enterprise will assist in analyzing system requirement and facilitating cooperation among the partners. This paper proposes to use strategic actor based on the i* framework to model virtual enterprise. The partners are treated as strategic agents and cooperation among them are depicted as dependency relationships. The actors in the model can form a hierarchical and federated architecture and cooperate with each other. So the virtual enterprise paradigm can be analyzed and realized based on the agent and goal-oriented approach. Example from an air-separator virtual enterprise is used to illustrate.

1 Introduction

A virtual enterprise is a dynamic alliance among companies. Through cooperating inside or outside companies, a virtual enterprise can develop some kind of products with shorter leadtime, better allover performance and lower cost to meet the rapidly changing market. Virtual enterprise is a temporary and virtual organization that is built to respond to the market more quickly and utilize various technologies and resources more efficiently in the entire virtual enterprise. It will be initiated by a core company which plays a role of leader. It will select proper cooperators to form a short-term relationship to collaborate in a one-off project. The initiator will select partners who can provide resources needed. In fact, a company can take part in different dynamic alliances and act as a different virtual group in different virtual enterprises. There are several cooperation styles, such as supply chain, subcontract process, joint venture management and so on [17]. The cooperation eventually maximizes the virtual enterpriseís utilization of resources, reduces high investment and risk, and ul-

P. Petta et al. (Eds.): ESAW 2002, LNAI 2577, pp. 270-283, 2003.

timately wins the competition. In order to form the short-term cooperative relationships with partners, the initial company must model the requirements of virtual enterprise and take on the high-level design task (product design task) as its goal. Then it can decompose the task into some sub-tasks. These partners are called the virtual groups who can take on the coarse-grained sub-tasks in accordance with their goals. They also can decompose their sub-tasks into the fine-grained sub-tasks and then select corresponding partners, and allocate these sub-tasks to them. So the whole high-level design task can be accomplished by these hierarchical partners. All the partners take on these sub-tasks are according to their goals and form a hierarchical alliance. It is very important to decompose the large and complex system into smaller, more manageable subsystems that can be dealt respectively.

Some architectures of virtual enterprises have been provided and many technologies have been applied in this field. Many sophisticated methodologies, such as CIMOSA, GRAI-GIM and PERA, are used to describe the lift-cycle of an integrated enterprise [16]. But these existing enterprise modelling methodologies are process-oriented approaches and relevant solutions should be predefined. Based on these modelling methods, the enterprise is represented as various function models and information models. In the meantime, they donít provide any tools and methods to identify the requirements of virtual enterprise. Because virtual enterprise is a dynamic alliance and the partners who are selected to join the alliance according to their goals maybe have different solutions to accomplish the same tasks or sub-tasks. So how to analyze the requirement of virtual enterprise is the foundation stone for modelling it. The partners who join the alliance will take on the tasks or sub-tasks and can be treated as agents which are capable of flexible, autonomous actions in virtual enterprise [2][10][15][20]. It will be very suitable to adopt an agent-oriented approach to analyze and model the requirements of a virtual enterprise and accomplish it in an agent-oriented architecture [3][4][11][12][13][14]. Adopting an agent-oriented approach to model virtual enterprise aims at support elicitation, exploration, and analysis of the systems-in-the-world and it can help users and stakeholders articulate what they concern and what cooperation relationships are between them [1][5].

In this paper, we adopt the i* framework to model a virtual enterprise. The i* framework [6] was developed for modelling intentional relationships among strategic actors, and helping to reason about changes in relationships among strategic actors. It adopts an agent-oriented perspective. In i* framework, organizations are viewed as consisting of social actors who have freedom of action (similar to agents), but depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished [7]. These dependencies are intentional and based on some important concepts such as goal, commitment, belief, capability and so on. It is very useful to provide a higher level modelling method comparing to conventional modelling techniques such as data flow diagramming and object-oriented analysis. From this viewpoint, the i* framework is similar to the Gaia methodology [11]. It also treats an organization as a collection of actors, but it more stresses on dependency relationships between the actors. It can support strategic actor to undertake the task according to its belief and capability in order to accomplish a goal. It also can reason about opportunities and vulnerabilities through all kinds of dependency relationships. This goal-oriented and agent-oriented analysis leads to the functional and non-functional requirements of the system-to-be. Based on the analysis, agent-oriented architectural design can be accomplished. This modelling approach will reduce a semantic gap between the agent-oriented software system and its implementary environment [8]. The framework has been applied in the context of requirements engineering, business processing reengineering, and other software processes. Now the framework is tending to form the basis of an agent-oriented system development paradigm.

In this paper, we explore the use of i* framework for modelling virtual enterprise. We can use organizational view to capture goals of the initial company. Goals related to functional capabilities provide the basis requirements of virtual enterprise and goals related to business and system qualities provide the non-functional requirements. The concept of softgoal is used to model quality attributes for which there are no *a priori*, clear-cut criteria for satisfaction, but are judged by actors as being sufficiently met. Goals from the organizational view take the first step to construct the virtual enterprise. They can be represented as a Strategic Dependency model in the i* framework. Then the partners who take on those goals will decompose the pertinent goals or tasks into alternative solutions and the softgoals will be used to determine how each alternative solution relates to pertinent business and system qualities. This can be represented as a Strategic Rationale model in the i* framework. Finally, the partners who take on the sub-goals and sub-tasks will refine the goal graph until the sub-goals and sub-tasks can be taken on by all hierarchical partners. The partners can be represented as actors in the i* framework.

This paper outlines an idea paradigm to use strategic actor relationships in the i* framework to represent the constructing and cooperative issue in a virtual enterprise, and to illustrate how these models can support analysis and reason. The modelling approach based on the i* framework can make a better understanding of the organizational relationships among those agents and the rationale of the decisions.

Section 2 presents a brief overview of the i* framework, including its basic concepts such as Strategic Dependency model and Strategic Rationale model used in the virtual enterprise. Section 3 describes the cooperation relationships modelling in the virtual enterprise and proposes a cooperative architecture to support it. Section 4 discusses related work.

2 The i* Framework for Modelling Virtual Enterprise

The i* framework consists of two kinds of models. One is the Strategic Dependency model that describes the network of intentional relationships among actors. Actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. These dependencies are intentional based on important concepts such as goal, commitment, capability, belief, and so on [6]. Another is the Strategic Rationale model which describes and supports the reasoning that each actor has about its relationships with other actors, its alternative means to achieve its goals, and how the qualitative expectations of actors are satisfied by these alternatives. Actors in i* framework are strategic because they can evaluate their social relationships in terms of opportunities that they offer, and vulnerabilities that they may bring.

2.1 Strategic Dependency Model

A Strategic Dependency (SD) model consists of many nodes and links. Each node represents an actor, and each link between two actors indicates that one actor depends on the other. The depending actor is called the depender and the actor who is depended on is called dependee. That means the depender needs some help to attain the goal from the dependee. The depender can achieve goals that otherwise cannot be realized without the dependency, or not as easily or not as effectively. At the same time, the depender becomes vulnerable. If the dependee fails, the depender would be adversely affected in its ability to achieve its goals. There are four dependencies in Strategic Dependency model: goal dependency, task dependency, resource dependency, and softgoal dependency. In a goal dependency, an actor depends on another to make a condition come true; In a task dependency, an actor depends on another to perform an activity; In a resource dependency, an actor depends on another for the availability of an entity; The softgoal dependency is a variant of the goal dependency and it is no *a priori*, clear-cut criterion for what constitutes meeting the goal. It reflects different types of freedom that is allowed in the relationship between depender and dependee[6].

Figure 1 show a Strategic Dependency model for the virtual enterprise. Here the virtual enterprise is constructed to produce a complex product, named as air-separator or oxygen-producer. It always needs a long time to produce an air-separator and it needs a lot of companies to cooperate with each other. The Product Designing and Quoting Group, Devices and Machines Manufacturing Group, Parts Supplying Group, and Product Assembling Group compose the virtual enterprise. These groups may consist of a set of related companies to finish respective goals. In fact, this forms a high-level cooperation in the virtual enterprise. The virtual enterprise is triggered by the requirement of market. In other words, the Customerís order is the requirement of the market. When the customer has a desire to purchase an air-separator, it is very difficult to get a product in stock, because this kind of product is customized and canít be produced in batch. So the customer must hand over the order to the Product Designing and Quoting Group in order to customize the product. The Product Designing and Quoting Group will design the product in accordance with the performance of product and give the quoted price and the delivering date. For the customer, it can evaluate the design, price and delivering date and decide to accept or not. It can get the related information, such as price and producing time from the Devices and Machines Manufacturing Group, Parts Supplying Group, and Product Assembling Group. If not, it can negotiate with the Product Designing and Quoting Group. If the negotiation fails, the virtual enterprise will not be built. The Product Designing and Quoting Group will design the product according to the performance of product. In order to enhance competition of the product, the Product Designing and Quoting Group should give the Customer a good design, including good performance, good price, and short delivering time. Good performance means the product should sufficiently meet the need of the product. It demands the Product Designing and Quoting Group to have sufficient experiences in product design. In the meantime, the Group should consider the related constraints in the downstream of producing process in advance, including the constraints in the manufacturing process, assembling process, and so on. It can avoid doing over again in order to short the delivering time and it also can effectively reduce the designing surplus in order to cut down the product price. All these can make the designed product more competitive. These constraints can be represented as all kinds of interaction relationships. So the Product Designing and Quoting Group is a footstone to build the virtual enterprise.

In figure 1, the Customer depends on the Product Designing and Quoting Group to get the product design scheme and evaluate the scheme, including the performance, price, and delivering time. The Product Designing and Quoting Group will depend on the Customer to get the product requirements. It also depends on the Devices and Machines Manufacturing Group to get the proper manufacturing information in order to cut down the designing surplus. It also can get sufficient information about the manufacturing price and time in order to give quoted price for the Customer. When the Product Designing and Quoting Group has designed the devices and machines it would negotiate with the Devices and Machines Manufacturing Group. When the performance, the price and the delivering date are decided, the cooperation among them will start. That will be a part of the virtual enterprise. In the meantime, the Devices and Machines Manufacturing Group will do its best to satisfy the Product Designing and Quoting Group in order to continue the business.

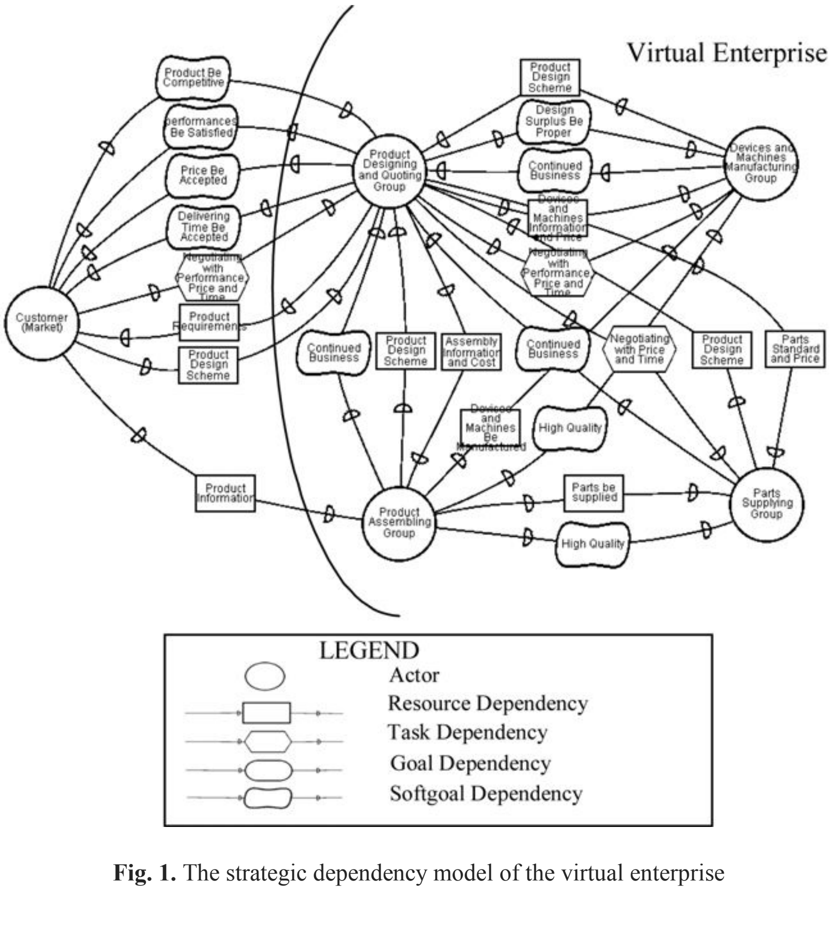


Fig. 1. The strategic dependency model of the virtual enterprise

Besides the cooperation above, the Product Designing and Quoting Group will cooperate with the Parts Supplying Group and Product Assembling Group. The Parts Supplying Group will purchase and provide all kinds of standard parts or instruments for the product. The Product Assembling Group will get all kinds of design information, devices and machines manufactured, and standard parts and then assemble them together. The product will be given a test run and a check, and will be accepted by the Product Designing and Quoting Group. It will be sent to the Customer by the Group and entire virtual enterprise will be dismissed.

2.2 Strategic Rationale Model

The Strategic Rationale (SR) model describes a more detailed level of an actor in order to model its internal intentional relationship and support to reason of each actor about its intentional relationships. The intentional elements, such as goals, tasks, resources and softgoals, appear in SR model not only as external dependencies, but also as internal elements arranged into a hierarchy of means-ends and task- decompositions relationships. A goal may be associated through means-ends with multiple, alternative ways to achieve it, which are usually represented as tasks. Task-decomposition links hierarchically decompose task into four types: sub-tasks, subgoals, resources, and softgoals [6].

In figure2, the goals of each group and internal intentional relationships are elaborated. The goal of the Customer is to get a customized product (air-separator) and is accomplished by ìRequirements of Product Be Decidedî. This task can be decomposed into three sub-tasks, such as ìPerformances Be Decidedî, ìPrice Be Decidedî, and ìDeadline Be Decidedî. For the Product Designing and Quoting Group, its goal is ìProduct Designing and Quotingî. There are three possible ways to accomplish it, such as ìNew_Product Designing and Quotingî, or ìTransformed_Product Designing and Quotingî, or ìModified_Product Designing and Quotingî. These tasks respectively response to different products and each task is composed of two sub-tasks, related product designing and quoting. In any designing way, there are five main designing processes, including ìGeneral Designingî, ìDevice Designingî, ìMachine Designingî, ìPower Supply System Designingî, and ìDetect System Designingî. In which, the Power Supply System includes many kinds of power transformers and the Detect System is used to detect flow quantity, pressure on the pipes, resistance in the pipes and so on. There are a lot of valves, detecting instruments and electronic controlling instruments. So these sub-tasks have different complexities and they have different quoted prices. The Product Designing and Quoting Group will select related cooperative partners whom the tasks and sub-tasks can be allocated to and roughly design the product according to the requirements from the Customer and give its quoting price. The Customer evaluates the design and price, and negotiates with the Product Designing and Quoting Group. When they come to an agreement, the cooperative partners will accept the order of product and make the detailed designs. The designing partners will consider the product manufacturing constraints and cost information from the partners in the downstream and the Product Designing and Quoting Group can negotiate with them to ensure the competitive price, delivering time, and good performance. Because the product is rather complex, the partners will decompose the designing work and find partners to accomplish it. These partners can cooperate with each other inside or outside their own companies.

In figure 2 shows the Strategic Rationale model to support reasoning about the product designing and quoting process. Each possible way has different implications for the qualities of goals or softgoals. A softgoal is usually not easy to quantified. The contributions to softgoals can be positive or negative, adequate or inadequate. The

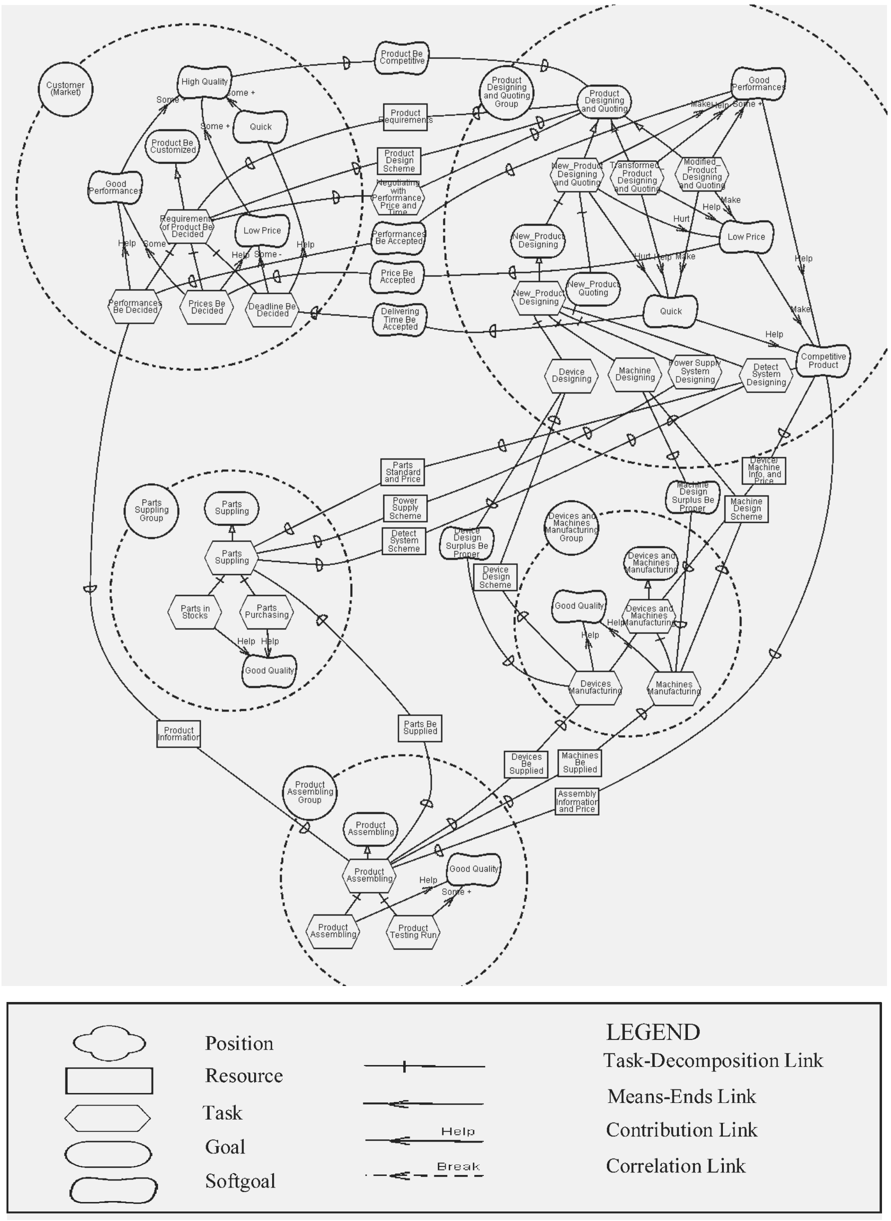


Fig. 2. The strategic rationale model of the virtual enterprise

treatment of softgoals is based on a framework developed by Lawrence Chung for dealing with non-functional requirements in software engineering[9]. The positive contribution types for softgoal are Help (positive but not sufficient), Make (positive and sufficient) and Some+(positive in unknown degree). The corresponding negative types are Hurt (negative but partial), Break (negative and sufficient) and Some-(negative in unknown degree). In figure 2, Modified_Product Designing is based on the existing designs and just need to do some little modifying work. So the designing process will be quick, low price, and have some positive effect to good performance; The Transformed_Product Designing is also based on the existing designs, but need to do a rather big transforming design. It will be some quick, some low price, and have some positive to good performance. These two processes are case-based designs and rather steady because there are already much experiences in the practices. But in the meantime they also have a difficulty to break through the constraints of those models. The New_Product Designing has an innovative designing method and may make sufficient positive effect to good performance, but it needs more time and high price. So the Strategic Rationale model provides a systematic way to explore the possible alternative in designing process. This would be help for the cooperators to decide which way they should take and sufficiently meet the requirements of the Customer.

In figure 2, there are still other Groups in the virtual enterprise. These groups supply related constraint information to the Product Designing and Quoting Group and cooperate with the Product Designing and Quoting Group to accomplish devices and machines manufacturing, parts supplying, and product assembling. In Devices and Machines Manufacturing Group, there are a lot of manufacturing experiences. The redundant surplus in design is the main reason for cost inflation, and sometime it is a factor to prolong the delivering time. So the control of surplus is very important in product design. But it must be large enough to ensure the product quality in order to satisfy the Product Designing and Quoting Group and continue their market in the future. The Parts Supplying Group is similar to the Devices and Machines Manufacturing Group. It has two ways to supply the parts, in stocks and purchasing. It also needs to provide good quality work to continue their cooperation. The Product Assembling Group will assemble the devices, machines and parts supplied by the groups mentioned above in accordance with the design of the product and will give the product a test run. Then the product will be checked and accepted by the Product Designing and Quoting Group and delivered to the Customer.

2.3 Roles, Positions and Agents

In i*, an agent is an actor used to refer to the concrete, physical manifestation; A role is an abstract and serves as holder of intention. It can be reasoned about before it is allocated to an agent. Position is typically assigned to an agent as a unit, serves as an intermediate mapping between agents and roles and eventually is occupied by some agents. Positions can cover roles, agents can occupy positions, and agent can also play roles directly [6]. The term actor is used to refer to the generic concept. Agent, role, and position are specializations of the actor concept.

In the virtual enterprise, the partners cooperate with each other as virtual groups according to their goals. In i*, goals always belong to agents and the global goal is distributed and propagated over the network of dependency relationships to form hierarchical intentions (goals). So an actor can represent an organization in which there are some other actors to take on the sub-tasks. Here the functional requirements are treated as goals and non-functional requirements are treated as softgoals which appear as dependency relationships among agents, like other goals.

In figure 3, those groups in figure 1 are represented as the high-level cooperators in the virtual enterprise and they are described as positions. These positions are occupied by some agents. For example, the position of Product Designing and Quoting Group covers two roles, Product Designing and Product Quoting. The Product Designing is so complex that it needs some partners to be accomplished. There are five partners in Product Designing and occupied by related agents who take on these decomposed tasks. Each position may cover different roles and be finished respectively.

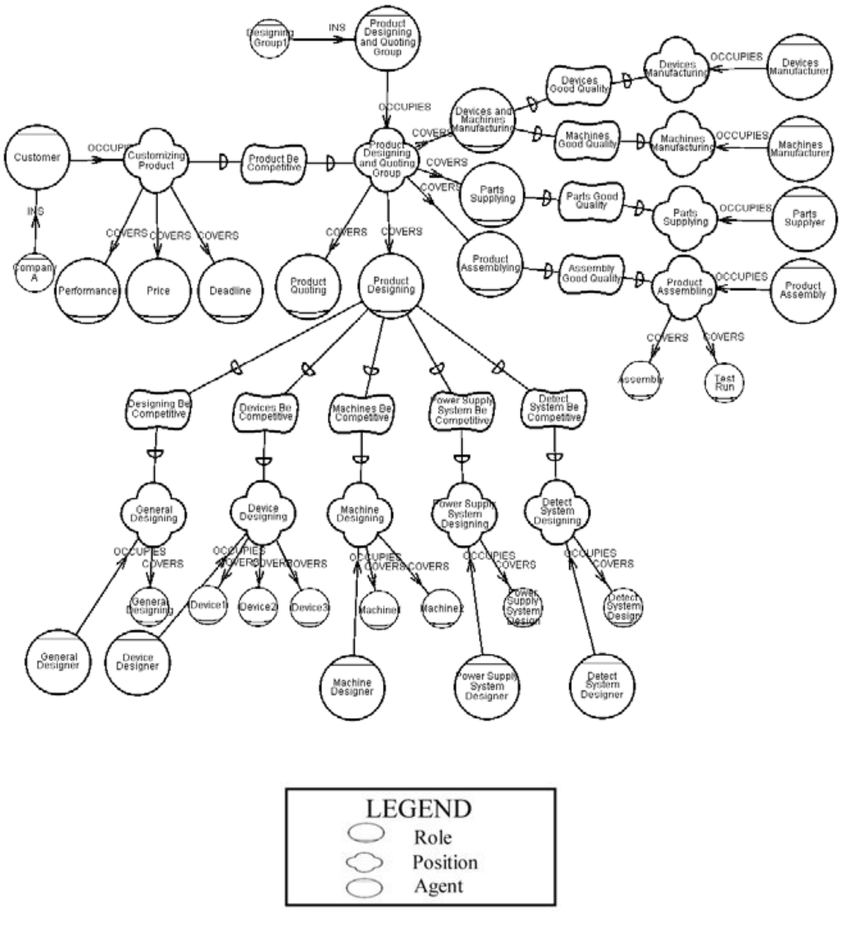


Fig. 3. The strategic dependency model with roles, positions and agents

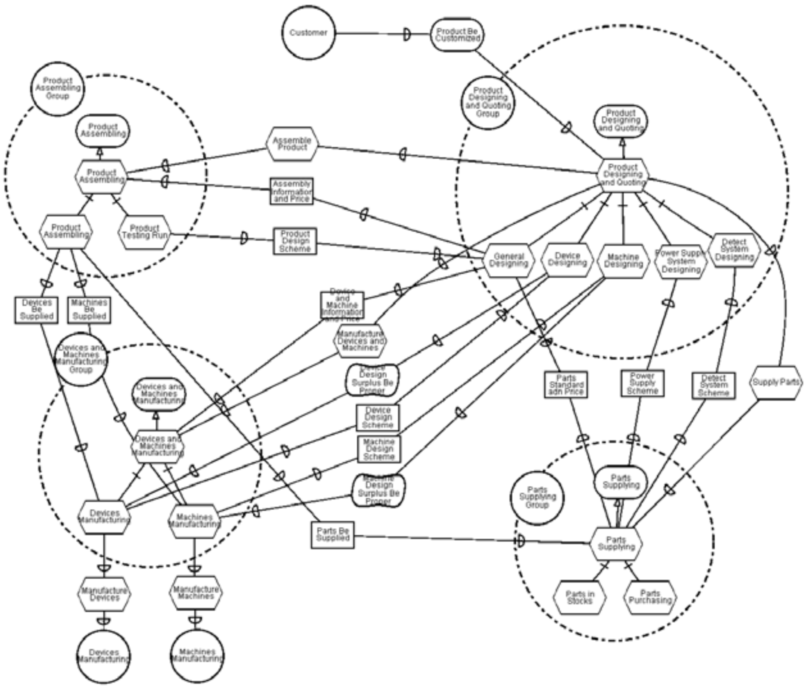


Fig. 4. The cooperation relationships in the virtual enterprise

3 Study of Cooperation in the Virtual Enterprise

3.1 Cooperation in the Virtual Enterprise

When the virtual enterprise is built, the partners will cooperate with each other according to their dependency relationships. The dependencies propagating over the network of dependency relationships make the dependers and dependees cooperate effectively.

In figure 4, the Customer depends on the Product Designing and Quoting Group (goal-dependency) to customize the product and doesnít care how the Group achieves the goal. In the virtual enterprise, the Product Designing and Quoting Group takes on a leader role to establish and organize the virtual enterprise in order to accomplish the task. The sub-tasks in the Product Designing and Quoting Group, such as General Designing, Device Designing, Machine Designing, will finish the general product design and all kinds of designs of devices and machines in this product. They respectively depend on Devices and Machines Manufacturing (task-dependency) to manufacture the devices and machines, depends on Parts Supplying (task-dependency) to supply the parts, depends on Product Assembling (task-dependency) to assemble the product.

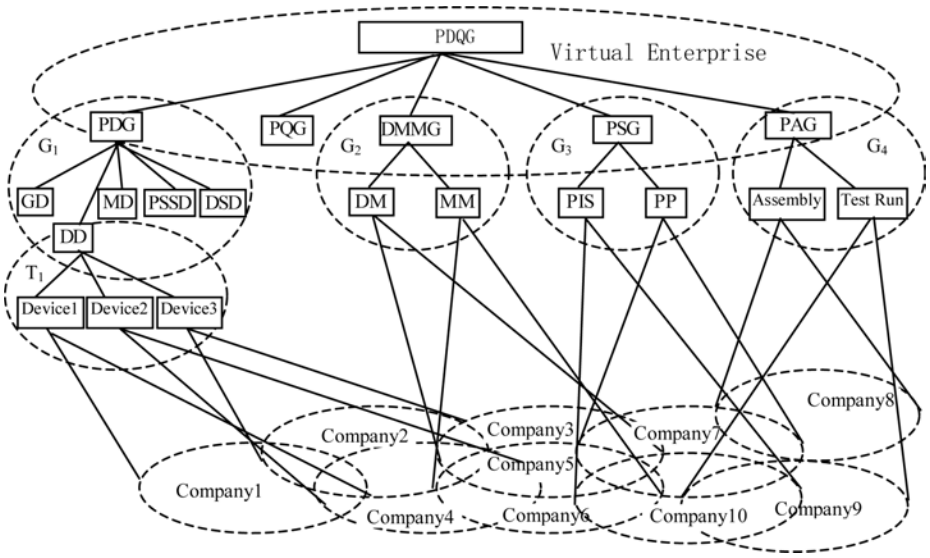
There are some resource-dependencies between tasks for the availabilities of entities (physical or informational). The Device Designing depends on the Devices Manufacturing (softgoal-dependency) to achieve proper surplus design for devices. The depender will make the final decision based on information that it get from these partners in the downstream in order to cut down the surplus properly. Beside the dependencies among the actors, there are many task decomposition links in the model that can describe the decomposition of tasks.

In figure 4, the sub-tasks of Devices Manufacturing and Machines Manufacturing are respectively allocated to two manufacturing partners. These two manufacturing partners can be selected by the Devices and Machines Manufacturing Group and can get those dependency relationships and related information from the their upper actor. From this view, the whole dependency relationships among the actors are formed a hierarchical architecture and can be constructed as a multi-agent system based on the federation architecture to support the virtual enterprise [15][20].

3.2 Cooperative Architecture for the Virtual Enterprise

Based on the analysis with the i* framework, an agent and goal-oriented model for the virtual enterprise can be built. Because the virtual enterprise is a dynamic alliance, when the goals and tasks have been allocated to partners, the allying process is finished. In order to support the allying process and cooperation in virtual enterprise, we propose a hierarchical and federated architecture in figure 5. In this architecture, the partners can communicate with each other with the well-known Contract Net Protocol [19] when they ally with each other. Those agents (not at the lowest level) can dynamically take two roles: manager or contractor [13]. As a manager, the agent will decompose the task or sub-tasks and allocate them to its partners. It can announce the task or sub-tasks to relevant partners and collect the bids from partners, and award the task or sub-tasks to the best bidders. When the task or sub-tasks have been allocated to the partners, the manager and the partners form a federation. The manager will cooperate and coordinate with the partners and support communication among the agents in its federation. Agents in a federation can communicate with others in different federations through their respective upper manager. This makes the realization of communication more convenient.

In figure 5, the hierarchical and federated architecture not only supports the task to be decomposed and allocated, but also to facilitate the tasks to be managed. In a federation, these partners cooperate with each other in order to accomplish the tasks or sub-tasks. These partners may come from different companies or come from different work units in the same company. They can overlap to implement their respective tasks or sub-tasks allocated to them based on various dependency relationships. That means those companies cooperating with each other in the virtual enterprise as different virtual groups, such as G_1, G_2, or G_3, respond different tasks or sub-tasks in the high-level cooperation. In the low-level cooperation, the developers and work units in same company or in different companies can form all kinds of work teams, such as T_1, to take on the decomposed tasks or sub-tasks. This makes the virtual enterprise more flexible and more competitive. In this paper, a partner can be a company, or a work unit, or a developer in a company.



Comments

PDQG:Product Designing and Quoting Group
DMMG:Device and Machine Manufacturing Group
PQG:Product Quoting Group
GD:General Designer
PSSD:Power Supply System Designer
DSD:Detect System Designer
MM:Machine Manufacturing
PP:Parts Purchasing

PSG:Parts Supplying Group
PAG:Product Assembling Group
PDG:Product Desinging Group
DD:Device Designer
MD:Machine Designer
DM:Device Manufacturing
PIS:Parts in Stocks

Fig. 5. A hierarchical and federated architecture for the virtual enterprise

4 Discussion

In this paper, we have illustrated the use of i* framework to model an air-separator virtual enterprise. The i* framework has been applied in many fields, but its applications are always in very concrete fields, such as trust-related issues, intellectual property management issues. The virtual enterprise is a very complex and distributed system and it is a dynamic alliance. The initial company establishes and organizes the virtual enterprise in order to sufficiently satisfy the requirements of the market. It will plan the product design and manufacturing process and select related partners according to their goals, intentions, and capabilities to join the dynamic alliance and share the task. It also needs to model the various dependency relationships among the partners in order to support their cooperation. So it is very important to model the virtual enterprise in order to reveal which parts are in it and how they interact and cooperate.

The i* framework was developed for modelling intentional relationships among strategic actors. The cooperators in the virtual enterprise are treated as actors and cooperate with each other in a network of dependency relationships. Some actors depend on goals and others depend on tasks or softgoals. Because some tasks are so complicated that they are decomposed into several sub-tasks and allocated to actors (partners) that have been selected to join the alliance. Here we specialize the actors who join the virtual enterprise as positions and assign the positions to agents who represent the partners. This realizes the task decomposition and task allocation. The i* model with the dependency relationships encourages and facilitates the analysis of requirements and cooperation in the virtual enterprise. We also propose a cooperative architecture based on this model to support virtual enterprise to be constructed and to be managed.

This paper has taken a rather simplistic view of the virtual enterprise even through we provide a complex alliance. We just give the coarse-grained analysis about it. It still needs rather deep study in the fine-grained level. In this paper, we model the virtual enterprise using the i* framework aiming at analyzing the requirements of virtual enterprise and cooperation among the partners. In future work, we plan to apply the i* framework to analyze and reason opportunities and vulnerabilities using the softgoals among the partners based on all kinds of dependency relationships. We also plan to combine other technologies, such as scenario notation [18], to achieve the development process from early requirement analysis to later detailed architecture design.

Acknowledgements

The work of this paper is greatly motivated and inspired by my cooperation with Professor John Mylopoulosís research. His kind support and encouragement is gratefully acknowledged.

References

[1] Bashar Nuseibeh, Steve Easterbrook. Requirements Engineering: A Roadmap. Proceedings of International Conference on Software Engineering (ICSE-2000), 4-11 June 2000, Limerick, Ireland, ACM Press.

[2] Carlos A. Iglesias, Mercedes Garijo and Jose C. Gonzalez. A Survey of Agent-Oriented Methodologies. Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98).

[3] Cutkosky, M.R., Engelmore, R.S., Fikes, R.E., Genesereth, M.R., Gruber,T.R., Mark, W.S., Tenenbaum, J.M. and Weber, J.C. (1993) PACT: An Experiment in Integrating Concurrent Engineering Systems, IEEE Computer, 26(1):28-37.

[4] Daniel Gross, Eric Yu. Evolving System Architecture to Meet Changing Business Goal: an Agent and Goal-Oriented Approach. ICSE-2001 Workshop: From Software Requirements to Architectures. May 2001, Toronto, Canada. pp. 13-21.

[5] Eric Yu. Agent-Oriented Modelling: Software Versus the World. Proceedings of Agent-Oriented Software Engineering AOSE-2001 Workshop, Montreal, Canada, May 29th 2001.

[6] Eric Yu. Modelling Strategic Relationships for Process Reengineering, Ph.D. thesis. Dept. of Computer Science, University of Toronto, 1995.

[7] Eric Yu and John Mylopoulos. Modelling Organization Issues for Enterprise Integration. Proceedings of International Conference on Enterprise Integration and Modelling Technology, October 28-30, 1997, Turin, Italy.

[8] Jaelson Castro, Manuel Kolp, and John Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems, 27(6), 365-389, 2002.

[9] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[10] Luis M. Camarinha-Matos and Hamideh Afsarmanesh. (ed.) Infrastructures for Virtual Enterprises-Networking Industrial Enterprises. Kluwer Academic Publishers. 1999.

[11] M. Wooldridge, N. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 3(3): 285-312, 2000.

[12] Nicholas R. Jennings. Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions. Artificial Intelligence, 75(2) 195-240, 1995.

[13] Nicholas R. Jennings. A Roadmap of Agent Research and Development. Int Journal of Autonomous Agents and Multi-Agent Systems, 1, 7-38(1998).

[14] Nicholas R. Jennings. On Agent-based Software Engineering. Artificial Intelligence, 117 (2) 277-296, 2000.

[15] Nicholas R. Jennings. An Agent-Based Approach for Building Complex Software Systems. Communications of The ACM, April 2001/Vol. 44, No. 4.

[16] Peter Bernus, Laszlo Nemes and Theodore J. Williams. (ed.) Architectures for Enterprise Integration. Chapman & Hall. 1996.

[17] Rick Dove. Agile Enterprise and Agile Production. Production Magazine, 1995.11.

[18] R.J.A. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. High Level, Multi-Agent Prototypes from a Scenario-Path Notation: A Feature-Interaction Example. In: H.S. Nwana and D.T. Ndumu (Eds), Third Conference on Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, 1998, p. 255-276.

[19] Smith R. G. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions on Computers, 1980, c-29 (12): 1104-1113.

[20] Weiming Shen, Douglas H. Norrie and Jean-Paul A. Barthes. (2001) Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing. Taylor & Francis.

Author Index