Michael Luck   Jorge J. Gomez-Sanz (Eds.)

Agent-Oriented Software Engineering IX

9th International Workshop, AOSE 2008
Estoril, Portugal, May 12-13, 2008
Revised Selected Papers

Springer

Volume Editors

Michael Luck
King's College London
Department of Computer Science
Strand, London WC2R 2LS, UK
E-mail: michael.luck@kcl.ac.uk

Jorge J. Gomez-Sanz
Universidad Complutense de Madrid
Facultad de Informatica
Avda. Complutense s/n, 28040 Madrid, Spain
E-mail: jjgomez@sip.ucm.es

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, I.2.11, F.3, D.1, C.2.4, D.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN	0302-9743
ISBN-10	3-642-01337-6 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-01337-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12649822 06/3180 5 4 3 2 1 0

Preface

Agent technology can be the key to a brand new family of applications providing outstanding features like autonomy and adaptation. However, the way in which such applications should be built is not yet clear to us. Just as with other kinds of software, we need expertise in the inherent problems, and we need to develop a substantial body of knowledge to enable others to take our efforts further. In the context of multi-agent systems, this knowledge is realized in the form of agent-oriented software engineering (AOSE). Thus, AOSE brings novel tools and methods with which developers can start to construct agent-oriented solutions for their problems.

AOSE has brought some important advances in agent research. These advances have been brought about because we tried to solve problems with what was available, and found that it was not sufficient. It is natural to conclude, therefore, that the more difficult the problems we consider, the more advances we will achieve.

Such advances necessarily mean research, and this is also consistent with the *engineering* spirit. AOSE provides a context in which formal and applied research meet, and it will remain so for a long time, and at the very least until AOSE matures to match object-oriented software engineering. For this to happen, again, increasing the complexity of our problems and showing the benefits of agent technology is required.

This complexity can come in different flavors. One such aspect is the problem size: can agent technology, as it is now, deal with development with a team of, say, eight people in a year? If verifiable evidence of this can be shown, it would provide a significant and welcome impetus to the area. With respect to the evaluation of the benefits to be gained, we have hypothesized for some time now that agent technology ought to find a natural niche in global computing, network enterprises, ubiquitous computing and sensor networks, to mention just a few examples. Undoubtedly, there is already functioning software in such domains, so the question arises as to why we should use agents instead. To address this concern, we need to demonstrate that an agent solution is better in at least one of the following aspects: it is more economical in cost, it is more robust and fails less often, it can be developed in less time, or it provides better performance.

The AOSE workshops aim to address these issues, both from a research perspective and from a perspective that is relevant to attracting the attention of developers. The issues all share a clear connection to the main problem: how to effectively and efficiently develop software systems using agent technologies. The AOSE workshops thus seek to contribute to the advance of AOSE by providing a forum for presentation, discussion and debate of exactly these concerns.

Building on the success of the eight previous workshops, the 9th International Workshop on Agent-Oriented Software Engineering (AOSE 2008) took place in Lisbon in May 2008 as part of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008).

The 2008 workshop received 50 submitted papers, a very strong number, which reflects the continuing interest in the area. All papers were reviewed by at least three reviewers from an international Program Committee of 42 members and 17 auxiliary reviewers, and presented papers were then subject to a second round of reviews for inclusion in this volume.

In structuring this volume, we have organized the papers into four sections. The first deals with multi-agent organizations, which provides a valuable abstraction for agents and needs appropriate integration into development methods. The second section addresses method engineering and software development processes, including papers that focus on the ways in which multi-agent systems are developed, the activities involved, the products generated, and the assessment of the suitability of methods for particular domain problems. The third section is dedicated to testing and debugging activities. Finally, the last section deals with tools and case studies.

1 Multi-agent Organizations

Starting with organizations, the first section begins with a paper by Coutinho et al., in which they propose an integration process for *organizational models* based on concepts from model-driven engineering (MDE). The process is applied to five organizational meta-models (AGR, $\mathcal{M}$OISE+,TÆMS, ISLANDER and OperA) to obtain an integrated meta-model, which can be used as an interlingua for multi-agent systems built according to any of the five organizational meta-models.

Continuing this idea, the second paper, by Argente et al., introduces a meta-model for organizations as an evolution of the INGENIAS meta-model. This new meta-model includes primitives to express norms, services, and a holonic approach for the definition of organizations.

Finally, the paper by Sudeikat et al. closes the section. It is concerned with the validation of the dynamics within an organization, and proposes a guide to validation based on simulations of the MAS under development. The paper illustrates the guide through validation of an intrusion-detection system.

2 Method Engineering and Software Development Processes

The second section begins with a paper by Seidita et al., aimed at grounding situational method engineering to show how to express an agent-oriented software engineering process using a software process engineering meta-model (SPEM). Situational method engineering promotes the idea that no single method can account for all methods needed by engineers, especially due to changes in the application domain. As an example, the paper illustrates the construction procedure with the formalization of the PASSI methodology, exploring which SPEM primitives are best suited for representing each part.

Also in the line of method engineering, García-Magariño et al. introduce the different issues arising during the specification of a development process for the INGENIAS methodology. First, they identify problems with existing process modeling tools, concretely EPF, APES, and Metameth. Then, they present a new tool, built with the Eclipse Modeling Framework, overcoming most of the difficulties, and providing an alternative implementation of SPEM. The validation of the tool is achieved using as a case study of some preliminary work undertaken on the formalization of the INGENIAS methodology development process. This leads to the identification of steps and artifacts that may benefit the formalization of other methodologies.

The next paper, by Rougemaille et al., starts with an introduction to SPEM and situational method engineering, before introducing the fragment concept using the domain of agent-oriented software engineering. Fragment implementation is achieved by means of SPEM, and requires the combination of several SPEM primitives. As a proof of concept, the paper provides some fragments identified in the ADELFE and PASSI methodologies.

Cossentino et al. introduce a procedure to build an ad hoc agent-oriented software engineering process using fragments from a library of methods extracted from different agent-oriented methodologies. These libraries account for pieces of MAS meta-models as well as the procedures used to instantiate these pieces. By using the libraries and the knowledge of the domain problem, a developer can then follow the procedure described in the paper to create an ad hoc methodology. The paper presents as proof of concept the ASPECS process, which is constructed out of fragments from PASSI, CRIO and Janus.

Following this, García-Magariño et al. present a set of metrics that serves to quantitatively evaluate the *availability* (whether a meta-model has all required elements to deal with a problem domain), *specificity* (accounting for concepts that are not used), and *expressiveness* (covering the number of elements that are needed to represent a system specification). These metrics are applied to six different meta-models from six different agent-oriented methodologies: Tropos, PASSI, Agile_PASSI, Prometheus, MaSE, and INGENIAS.

In a different vein, the work by Padgham et al. proposes a new notation that may facilitate communication among different agent-oriented software engineers. Although meta-models of different agent-oriented methodologies remain unchanged, the use of a common representation of concepts can make these models appear to be more similar. Here, the semantics are still different so that an agent in PASSI is still different from an agent in MaSE. The effectiveness of the solution is still to be evaluated, but it provides an alternative to the unification of existing meta-models.

Continuing the methodology integration topic, Gascueña et al. show that it is not necessary to build a new methodology from fragments or to unify meta-models to gain the benefit of the different methodologies: their paper describes how to combine INGENIAS and Prometheus while keeping the benefits of both. The idea is to use each methodology only at concrete points in the development process, when a transition from Prometheus to INGENIAS is required. Since this transition suggests a manual translation of Prometheus concepts into INGENIAS ones, the authors identify such a mapping, and describe how it can be accomplished by using concrete examples.

By contrast, Hahn et al. criticize the meta-model approach, due to limitations in expressing semantics, and instead propose specifying semantics with Object-Z, a state-based and object-oriented specification language. Their paper introduces the basics of Object-Z and explains how it can be used to add semantics to a concrete meta-model called DSML4MAS, claiming that this can also be applied to other existing meta-models. In particular, they argue that the Object-Z semantics can be partially translated in terms of the Object Constraint Language (OCL), a language traditionally used together with meta-models.

The section concludes with a paper from Dam et al., dealing with two important problems in a software system: its maintenance and evolution. The approach is based on a library of repair plans to fix inconsistencies in the design model, a meta-model of the problem, and consistency constraints. When a change is requested, the system analyzes the current specification and then chooses a repair plan to modify the specification. The procedure is implemented with the Change Propagation Assistant (CPA), a prototype system.

3 Testing and Debugging

The section on testing and debugging, an important area that brings AOSE closer to the concerns of real-world commercial deployment, commences with a paper by Ekinci et al. The paper suggests performing testing activities by using goals as the driving criteria, and defines the concept of *test goal*. This concept represents the group of tests needed in order to check if a goal is achieved correctly. Here, each of the needed tests is considered to have its own goal to check, so that a *test goal* has three subgoals: setup (prepare the system); goal under test (perform actions related to the goal); and assertion goal (check goal satisfaction). These ideas are implemented in the SEAUnit test tool.

Looking for other kinds of driving criteria in testing, Nguyen et al. propose using the ontologies extracted from the MAS under test and a set of OCL constraints, which act as a test oracle. Having as input a representation of the ontologies used, the idea is to construct an agent able to deliver messages whose content is inspired by these ontologies. The resulting behaviors are regarded as correct using the input set of OCL constraints: if the message content satisfies the constraints, the message is correct. The procedure is supported by *eCAT*, a software tool.

The third paper in the section, by Gomez-Sanz et al., describes progress made in the INGENIAS methodology in these areas. With respect to testing, the INGENIAS meta-model is extended with concepts for defining tests, and the code generation facilities are augmented to produce JUnit skeletons based on these definitions. With respect to debugging, the system is integrated with ACLAnalyzer, a data-mining facility for capturing agent communications and exploring them with different graphical representations. The paper finishes with a survey and categorization of different work on testing and debugging in the agent literature.

4 Tools and Case Studies

In the last section of the book, we focus on tools and case studies, seeking to provide valuable experience reports and practical assistance. The first paper of this section, by Cabrera-Paniagua et al., is an application paper describing development experience with the PASSI methodology. The chosen case study concerns a system simulating a passenger transportation enterprise within which agents represent different transport companies trying to satisfy the needs of simulated users.

The second paper, by Nunes et al., is also an application paper with different versions of a conference management system. Its focus lies in the analysis and comparison of the evolution from a non-agent-oriented system to a product-line agent-oriented system. The paper ends with an interesting evaluation of the different variability types identified in the agent system, as well as a discussion of how refactoring of the system could have been done in other ways.

The paper by Yoo et al., introduces a tool that combines JADE and Repast in order to provide the construction of simulations. This combination seems well suited for working with value-adding networks, which are complex networks of partners arranged by an enterprise. The use of Repast enables the problem to be modeled quickly, while the simulation itself is achieved by using JADE.

Following on from this, van Putten et al. present another tool resulting from the combination of OperA and Brahms, the former being a modeling language for MAS, and the latter a development environment for MAS based on the concept of activity. A simulation of air traffic is the chosen case study where this combination is put to the test.

Finally, the section (and the book) ends with a paper by Gorodetsky et al., presenting a support tool that can be used with the Gaia methodology, although the paper also provides additional guidelines to deal with the design and implementation stages. The tool enables the generation of executable code and implements non-trivial parts of Gaia, such as liveness expressions, and the paper provides examples of the tool applied to an air traffic management case study.

November 2008

Jorge J. Gomez-Sanz
Michael Luck

Organization

Workshop Chairs

Michael Luck (Co-chair)
Department of Computer Science
King's College London
UK
Email: `michael.luck@kcl.ac.uk`

Jorge J. Gomez-Sanz (Co-chair)
Facultad de Informatica
Universidad Complutense de Madrid
Spain
Email: `jjgomez@sip.ucm.es`

Steering Committee

Paolo Ciancarini	University of Bologna, Italy
Michael Wooldridge	University of Liverpool, UK
Jörg Müller	Technische Universität Clausthal, Germany
Gerhard Weiss	Software Competence Center Hagenberg GmbH, Austria

Program Committee

Claudio Bartolini (USA)
Bernard Bauer (Germany)
Federico Bergenti (Italy)
Carole Bernon (France)
Olivier Boissier (France)
Paolo Cianciarini (Italy)
Massimo Cossentino (Italy)
Keith Decker (USA)
Scott DeLoach (USA)
Noura Faci (UK)
Klaus Fischer (Germany)
Rubén Fuentes (Spain)
Paolo Giorgini (Italy)
Marie-Pierre Gleizes (France)
Nathan Griffiths (UK)
Michael Huhns (USA)
Vicent Julian Inglada (Spain)
Joao Leite (Portugal)
Jürgen Lind (Germany)
Carlos Jose Pereira de Lucena (Brazil)
Viviana Mascardi (Italy)
Simon Miles (UK)
Sanjay Modgil (UK)
Haris Mouratidis (UK)
Eugenio Oliveira (Portugal)
Andrea Omicini (Italy)
Nir Oren (UK)
Juan Pavón (Spain)
Michal Pechoucek (Czech Republic)
Joaquín Peña (Spain)
Anna Perini (Italy)
Eric Platon (Japan)

Alessandro Ricci (Italy)	Arnon Sturm (Israel)
Fariba Sadri (UK)	Laszlo Varga (Hungary)
Brian Henderson-Sellers (Australia)	Michael Winikoff (Australia)
Onn Shehory (Israel)	Eric Yu (Canada)
Viviane Torres da Silva (Spain)	

Auxiliary Reviewers

Estefania Argente	Cristián Madrigal-Mora	Cristina Ribeiro
Petr Benda	Frederic Migeon	Luca Sabatucci
António Castro	Ambra Molesini	Valeria Seidita
Adriana Giret	Mirko Morandini	John Thangarajah
Christian Hahn	Elena Nardini	Jiri Vokrinek
Sachin Kamboj	Cu Nguyen Duy	

Table of Contents

Model-Driven Integration of Organizational Models

Luciano R. Coutinho[1,⋆], Anarosa A.F. Brandão[1,⋆⋆], Jaime S. Sichman[1,⋆⋆⋆,†], and Olivier Boissier[2,†]

[1] LTI / EP / USP - Av. Prof. Luciano Gualberto, 158, trav. 3
05508-900 São Paulo, SP, Brazil
{luciano.coutinho,anarosa.brandao,jaime.sichman}@poli.usp.br
[2] SMA / G2I / ENSM.SE - 158 Cours Fauriel
42023 Saint-Etienne Cedex, France
Olivier.Boissier@emse.fr

Abstract. Currently, the design and running of a Multi-Agent System (MAS) mobilize several models. Besides agents' architectures, high level agent communication languages and domain ontologies, explicit *organization specifications* (written in some *organizational model*) are more and more used to structure and constrain the behavior of MASs. In the case of open MASs, one important requirement is interoperability w.r.t. these models. Focusing organizational model interoperability, in this paper, we propose an integration process for *organizational models* based on concepts (*models*, *mappings*, *transformations*, etc.) and techniques (*Match*, *Merge*, *TransfGen*, *Compose*, etc.) from Model-Driven Engineering (MDE). The process is concretely used to integrate five organizational models: AGR, $\mathcal{M}$OISE+, TÆMS, ISLANDER and OperA.

1 Introduction

An *organizational model* is a conceptual framework together with some modeling language used to specify the *organization* of Multi-Agent Systems (MASs). In general, we can say that the organization of a MAS comprises stable patterns or structures of joint activity that constrain and drive the actions and interactions of agents towards some desired global purpose. An organizational model can be used both at design and run time of a MAS. At design time, it is a specification technique to describe the MAS architecture [1,2,3,4,5]. At run time, there are some platforms and middlewares [6,7,8] that interpret organizational specifications and instantiate components that explicitly embody and enforce the organization within a (running) MAS. In these implementations, one interesting feature is that agents have the possibility of querying the organization specification and to reason about it. This enables the construction of highly open and effective MASs: systems where *external agents* can enter and leave at their will without disrupting the organization imposed on the MAS.

In a previous work [9], by reviewing eleven organizational models and variations, we have identified four main dimensions in the modeling of MAS organizations: the

⋆ Supported by FAPEMA, Brazil, grant 127/04 and CAPES, Brazil, grant 1511/06-8.
⋆⋆ Supported by CNPq, Brazil, grant 310087/2006-6.
⋆⋆⋆ Partially supported by CNPq, Brazil, grants 304605/2004-2, 482019/2004-2, 506881/2004-0.
† Partially supported by USP-COFECUB, grant 98/-4.

structural, *functional*, *interactive* and *normative* dimensions. The structural dimension deals with the static structural aspects of any organization (e.g. roles, role relations and groups). The functional dimension expresses the global goals of the organization and their decomposition into sub-goals. The interactive dimension specifies dynamical interactions structures connecting roles. Finally, the normative dimension provides norms and rules that interconnect elements of the other dimensions.

Now, building upon our previous results, we propose an integration of organizational models. The main motivation behind our research effort is to promote *organization interoperability* in open MASs. In such systems, agents should be able of interacting using a high level communication language; they should also "understand" the concepts or ontology of a given application domain; and in the case of MASs with explicit organizations, as we discussed above, they should abide by structures, constraints and norms existent in the MASs. If we plan to port an existing MAS from one platform to another or, more ambitiously, if we want to construct federations of interconnected organized heterogeneous MASs then we will need some interoperability strategy in place. In spite of this, while interoperability w.r.t. communication languages and domain ontologies have been studied [10,11], the organizational interoperability problem has not been explicitly addressed in the literature.

Thus, in this paper we propose a general process for the integration of organization models that can be used to address problems of organizational interoperability. Concretely, the process is illustrated by the integration of five organizational models from the literature: AGR [1], TÆMS [2], $\mathcal{M}$OISE+ [3], ISLANDER [4] and OperA [5]. These models are compared using an iterative and incremental process guided by the modeling dimensions discussed above. In this process we *match* and *merge* each organizational model with the previous integration result. The result, the final integrated model, is a *duplicate-free union* of the several models. It can be used to: automatically synthesize *transformations* between organizational specifications; design agent architectures that take into account a broader picture of organization; or as a first step towards a possible organizational model standard.

In the above account, the terms *match*, *merge*, *duplicate-free union* and *transformations* were taken from the *model management vision* [12,13], which is currently treated as part of the emergent Model-Driven Engineering (MDE) research field [14]. The use and adaptation of principles and technology from MDE to the MAS domain is the second contribution of this paper. The basic idea of MDE is the use of the *model* concept as a unifying principle along all the phases of a software engineering enterprise [15]. The model concept is thus enlarged to encompass any formal representation that is conformant to *metamodels*. A metamodel is itself a model that defines a modeling language (a set of well formed models). In this unified sense, not only analysis and design models are deemed as models, but also artifacts like code, language definitions (metamodels) and transformations definitions among models in different levels. In particular, the model management vision defines a set of general operators (*Match*, *Merge*, *Compose*, etc.) to deal with common perceived model related activities. By using an MDE approach, we gain both in the reuse of software modeling frameworks, specially those that automatize operations on models, and in providing a test case for the MDE community in the area of organization centered MAS engineering.

The paper is divided into seven sections. In section 2, we present a motivating example in the domain of health care. In section 3, we discuss the concepts and operators comprising the model management algebra. In section 4, the main process for organizational models integration is presented. The integration of AGR, $\mathcal{M}$OISE+ and TÆMS is described in detail. The main results of integrating ISLANDER and OperA are briefly commented. In section 5, we present the implementation and use of an integrated organizational model. In section 6, we relate our work to others in the literature. Finally, in section 7 we present our conclusions and future work.

2 Motivation

In order to put in concrete terms the need for an integrated view of organization in the engineering of MAS, we will briefly describe an application scenario. The scenario is an innovative vision for health care information system [16]. The main idea is to shift the focus from information systems based on the care provider (hospitals, clinics, etc) to one based on the individual:

> "Such a system, which we call 'Guardian Angel' (GA), integrates over a lifetime all health-related information about an individual (its 'subject'), thus providing, at minimum, a comprehensive medical record that is often virtually impossible to reconstruct in a timely manner as the subject moves through life and work assignments. But GA is to be not merely a passive repository of information, but an active process that: (a) engages in data collection, ... (f) contains ... subjects' preferences, represents these in a broad range of negotiations with other systems, including setting therapeutic guidelines and scheduling appointments, ... (j) provides patient support functions such as contacts with support groups and other patients, queries to pharmaceutical companies, government agencies, etc. " [16, p. 3]

By its characteristics, the GAs can be conceived as agents and the systems it interacts with (systems of the care providers, support groups, etc.) as open organized MASs. In an ideal world, the MASs are conceived using one widely acceptable and comprehensive organizational model. In such world, the GA designer concern about organization interoperation is to build agents that interpret specifications written in a given predefined organizational model.

In the actual state of the art, however, the situation is different. There is not a standard comprehensive organizational model. Facing heterogeneous models, one solution is to require that the MAS designers provide representative agents for the GA inside the MAS that "understand" the organization specifications and know how to act in accordance to it. This solution turns the MAS into semi-closed societies [17] were external agents loose their organizational autonomy. Another solution is to provider an organization interoperability layer between the GAs and the MASs. One service provided by this interoperability layer would be to translate among organizational models. In the case of compatible organizational models (i.e., that address the same modeling dimensions [9]), this is a feasible solution. A third, and more ambitious solution, is to take the several organizational models behind the systems the GAs have to inferface with and to produce an agent that is able to specialize its organizational reasoning to each model as needed.

Both the second (interoperability layer) and third (agent that reasons globally) solutions can be operationalized by means of some integration of organizational models. Regarding the second solution, the advantage of an integrated approach is to avoid a great number of translations between organizational models. In the case of the third solution, we can avoid redundancies in the agent architecture. However, in these solutions, one challenge - the one that is explicitly addressed in this paper - is how to meaningfully integrate the models and maintain this integrated view as new models come into scene.

In the following we describe the *model management algebra*, a set of model operators that provides us with a suitable formalism for describing and solving the problem of meaningfully doing and maintaining an integration of organizational models.

3 Model Management Algebra

In MDE, a Model Management System is a tool that helps designers to perform (ideally in an automatic way) common operations over models and mappings between models [12,13]. Among these are the *Match*, *Merge, TransfGen* and *Compose* operators.

3.1 Models and Mappings

From this point, we will use the term 'model' in a strict technical acception to mean a *set of objects* (elements with identity), usually forming a *graph structure* with distinct *node* and *edge* objects, *conforming to* a given *metamodel* and *representing* some entity or process in the development of a software solution. It is out of the scope of this paper to provide a detailed account of the conformance and representation notions (we point the reader to [15]). By its turn, the term 'metamodel' will be used to mean a *model that defines a set of structurely well formed models*. A set of models we will call a *modeling language*.

Given the above distinction, we can say that the 'organizational model' of the previous sections encompasses a metamodel for an organization modeling language (an *organization metamodel*). And, 'organization specification' is a model that conforms to a given organization metamodel.

Let M_1 and M_2 be models with graph structures represented by triples (N_i, E_i, e_i), $i = 1, 2$, where N_i is their set of nodes, E_i is their set of edges and $e_i : E_i \rightarrow N_i \times N_i$ is the 'edge function' that associates each edge with a pair of nodes of M_i. A mapping model map_{12} between M_1 and M_2 is a model M, represented by a binary relation $\{(o_1, o_2) | o_i \subseteq N_i \cup E_i,\ i = 1, 2\}$. Such mapping reifies the concept of relationships between models.

3.2 Operators

The model management operators take models and mappings as input and produces models and mappings as output [18]. In this paper, we are interested in the operators:

- $Match(M_1/\mu_a,\ M_2/\mu_b) \rightarrow map_{12}/\mu_m$: this operator takes two models M_1 and M_2, respectively conformant to metamodels μ_a and μ_b, and produces the mapping map_{12} (between M_1 and M_2), which conforms to the metamodel μ_m. Objects in the input models that are related by a pair in map_{12} are considered either *equal* or *similar*. *Match* has been implemented as a semi-automatic operator [19].

- $Merge(M_1/\mu_a,\ M_2/\mu_b,\ map_{12}/\mu_m) \rightarrow (M_3/\mu_c,\ map_{13}/\mu_n,\ map_{23}/\mu_n)$: this operator takes two models M_1 and M_2, and their associated mapping model map_{12}, and produces a third model M_3 and two new mapping models map_{13} and map_{23}. The resulting model M_3 is a *duplicate-free union* of M_1 and M_2. That is, it includes a copy of all objects of M_1 and M_2, except those declared equal through map_{12}. The equal objects are collapsed in a single object in M_3 that contains their properties and relationships. The mappings map_{13} and map_{23} relate each object of M_3 to the objects it was derived from. We refer the reader to [20,18] for a detailed description of *Merge*.
- $TransfGen(\mu_a,\ \mu_b,\ map_{ab}/\mu_m) \rightarrow Transf_{ab}$ - this operator takes two metamodels μ_a and μ_b and produces as output a transformation $Transf_{ab}$ that translates models that are conforming to μ_a into correspondent models conformant to μ_b. The correspondence relationships between μ_a e μ_b are defined in the input mapping map_{ab}. The output $Transf_{ab}$ has the following general form:

$$Transf_{ab}(M_1(s)/\mu_a) \rightarrow M_2(s)/\mu_b$$

  where the argument (s) is used to indicate that both M_1 and M_2 are correspondent models, written in different metamodels, of the same subject.
- $Compose(map_{12}/\mu_a, map_{23}/\mu_b) \rightarrow map_{13}/\mu_m$: this operator takes two mapping models and produces a new mapping model that is the composition of the previous ones (i.e., $map_{13} \equiv map_{23} \circ map_{12}$).

3.3 Scripts

A *script* is a sequence of consecutive applications of the model management operators. For example, suppose that we have n organization metamodels μ_i and we want to build transformations that translate specifications from one to another. A simple script that satisfies this task is: $for\ \ 1 \leq i, j \leq n,\ \ do\ TransfGen(\mu_i, \mu_j, Match(\mu_i, \mu_j))$.

Another script for the same task is: do *Match* and *Merge* μ_1 with μ_2 producing the first integrated metamodel μ_1^{int}; then, *Match* and *Merge* μ_3 with μ_1^{int}producing the second μ_2^{int}; and so on, until *Match* and *Merge* μ_n with μ_{n-2}^{int} producing the last μ_{n-1}^{int}. Finally, do $TransfGen(\mu_i, \mu_j, Compose(Match(\mu_i, \mu_{n-1}^{int}), Match(\mu_{n-1}^{int}, \mu_j)))$ to obtain a particular transformation.

Comparing the scripts, the first involves the operators *Match* and *TransfGen* while the second uses four operators. However, the computational effort associated with the second script tends to be lower due to the number of times the operator *Match* (which is difficult to automate because it involves interpreting one model against the other) is executed. Regarding the results, if we consistently apply the same criteria to perform the *Match*, both scripts are expected to produce similar transformations among the metamodels. This is due to the fact that the *Merge* operator preserves all the information producing a duplicate-free union of the source metamodels.

3.4 Correspondence Metamodel

Match, *Merge* and other operators depend on mapping models. Mapping models conform to some *correspondence metamodel*, a model that defines a mapping language.

Several correspondence metamodels proposals are found in the literature [21,18,22]. Among these, we have adopted the *Atlas Model Weaver core metamodel* (AMW)[22]. This is a class-based metamodel that defines abstract classes that represent *links*, *link ends* and *element identifications*. A link denotes a correspondence relation between modeling elements. A link can have one or several link ends, each end having the identification of the element referenced by the link. A link can also have nested child links.

Three features of AMW have influenced our choice: (*i*) it is a simple metamodel that presents the basic characteristics of the other candidates;(*ii*) it proposes an extension mechanism that can be used to define our own concrete correspondence links; and (*iii*) it is implemented and freely available for use as part of a model management system - the AMMA[1] - that runs over Eclipse platform[2].

Using the extension mechanism of AMW, we have extended the core metamodel with the automatic running of a generic *Merge* algorithm [20]. This extension is based on the use of two basic concrete links: *equality* and *similarity*. Elements linked by an equality link have to be collapsed in one during a merge. Elements linked by a similarity link have some semantic relationship but should not be collapsed in one during a merge. For instance, let us consider o_1 model element from model M_1 linked to o_2 model element from model M_2 by an equality link. During a merge both have to be replaced by o_u, unique element which gathers all the properties of o_1 and o_2 . If o_1 has an attribute id and o_2 has an attribute $name$ that have the same meaning in M_1 and M_2 (e.g. unicity), they can be equated. But, if one attribute has to be unique and the other not, it is better to link the two attributes with a similarity link instead of an equality link.

4 Integration of Organizational Models

Given the general description of the previous sections, we now present the process we have defined to integrate organization metamodels. In our experiments, we have worked with the metamodels of five organizational models: AGR, $\mathcal{M}$OISE+, TÆMS, ISLANDER and OperA (see [9] for a presentation and comparison of these models). In the sequel, we will illustrate the integration process for the first three models. We will focus mainly on the application of *Match* and *Merge* to achieve an integration.

4.1 General Process

We follow an iterative process that is similar to the second script discussed in the section 3.3. In each iteration (except the first), we deal with an organizational metamodel and a current integrated metamodel. We *Match* the organizational metamodel against the current integration; after, we perform a *Merge*. At the end, we have a new integrated metamodel and a mapping from the metamodel to the new integrated metamodel.

Match. In this process the more challenging step is to perform the *Match* operation. As we have commented in section 3.3, the *Match* operation is rather difficult to automate because it involves interpreting one model against the other. In our specific case, we

[1] http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/
[2] http://www.eclipse.org

deal with the challenge of finding correspondences between the metamodels primarily in a manual basis.

Firstly, we divide the work according to the modeling dimensions we have proposed in [9]. In this sense, we hold the premise that any system model (not only organizational models) will provide modeling constructs in some (or all) of the following categories:

– *functional dimension* — modeling constructs to represent the functional aspects (functions, function decompositions) of the system and its sub-systems;
– *structural dimension* — modeling constructs to lay down the time-invariant structure (basic components and their relations) of the system;
– *interactive dimension* — modeling constructs to represent the actions/interactions occurring through time that respect (or produce) the time-invariant structure and that realizes the functional aspects of the systems.

In the case of agent organizations, where the components are capable of performing autonomous actions, we add a forth category of modeling constructs:

– *normative dimension* - modeling constructs to flexibly couple agents to the functional, structural and interactive aspects of organizations; where flexibly means trying to reach a balance between the agent autonomy and the organizational purposes.

Secondly, we compare the general structure of the metamodel (in a given modeling dimension) against some recurring structures. In the case of functional elements, the general notion is that of goal (function, or objective) and their structuring (decomposition) in goal-trees (plans). Regarding structural elements, we search for roles, roles structures, groups and group structures. The common interactive elements are interaction protocols and networks of protocols. Finally, the basic normative elements are norms and rules.

Lastly, we take into account specific structural similarities between the metamodels. Here, we check what modeling constructs can be equated by putting into correspondence sub-sets of their attributes and associations to other constructs. To a certain degree, this check can be performed in an automatic manner by using some tools reported in the literature [19,21].

Merge. We merge organization metamodels using a detailed algorithm [20] implementing the *Merge* semantics described in section 3.2. The algorithm tries to satisfy the following Generic Merge Requirements (GMRs) (in the sequel, *M* denotes the result of *Merge*):

– *element preservation* - each element in the input has a corresponding element in *M*;
– *equality preservation* - input elements are mapped to the same element in *M* iff they are equal in the mapping;
– *relationship preservation* - each input relationship is explicitly in or implied by *M;*
– *similarity preservation* - elements that are declared similar (but not equal) to one another retain their separate identity in *M* and are related to each other by some relationship;
– *meta-metamodel constraint satisfaction* - *M* satisfies all constraints of the meta-metamodel (the language used to write the metamodels);

- *extraneous item prohibition* - no additional elements or relationships other than the above exist in *M*;
- *property preservation* - for each element *e* of *M*, *e* has property *p* iff *p* is a property of an element *o* of the input models and *e* is merged with *o* by the above.

4.2 Iteration 1

In the first iteration, we do not a have a current integrated organization metamodel. Thus, we deal with two particular metamodels and produce the first integration. We take AGR and $\mathcal{M}$OISE+. In order to match AGR and $\mathcal{M}$OISE+, we first notice that the modeling constructs found in AGR are situated in the structural dimension. On the other hand, the modeling constructs in $\mathcal{M}$OISE+ spread the structural, functional and normative dimensions. In this way, the match will be restricted to structural elements.

Match of AGR and $\mathcal{M}$OISE+. The result of matching AGR and $\mathcal{M}$OISE+ is the mapping model illustrated in Fig. 1. The mapping model conforms to the correspondence metamodel from sub-section 3.4 (AMW extension). Fig. 1 shows that AGR and $\mathcal{M}$OISE+ have three main correspondence clusters.

The first cluster is among classes: `AGR!OrgStructure`, `MOISE+!OrganizationalSpec` and `MOISE!StructuralSpec`. They are the root of an organization specification. The correspondences (rounded rectangles) say that `AGR!OrgStructure` corresponds directly to `MOISE+!OrganizationalSpec`, and indirectly (by means of the reference `gs:AGR!GroupStructure`) to `MOISE+!StructuralSpec`.

In the second cluster we have `AGR!GroupStructure` in correspondence to `MOISE+!GroupSpec`. Both are used to specify groups inside organizations. Here, `AGR!GroupStructure` is composed of `AGR!Role` and `MOISE+!GroupSpec` is composed of `MOISE+! GroupRole`. This gives us a hint that `AGR!Role` has to be made equivalent to `MOISE+! GroupRole` in a third cluster.

Besides an identification, `AGR!Role` has information about the minimum and maximum number of agents allowed to play a role. `MOISE+!GroupRole` has this same information but in a different structure. It has a reference to `MOISE+!Cardinality` that holds the maximum and minimum information. It has a reference to `MOISE+!Role` that contains a role identification.

Merge of AGR and $\mathcal{M}$OISE+. Having in mind the GMRs (section 4.1) and using the mapping of the previous section, we have produced the *Merge* of AGR and $\mathcal{M}$OISE+ that is illustrated in Fig. 2. In the figure, we have classes for the correspondences between AGR and $\mathcal{M}$OISE+ (`OrgSpec`, `GrSSpec` and `GrRole`); the other classes (and their respective attributes and relationships) appear only in $\mathcal{M}$OISE+ or in AGR. The classes `FunctionalSpec` and `DeonticSpec` appears only in $\mathcal{M}$OISE+ and are the root for other modeling constructs situated in the functional and normative dimensions, respectively.

4.3 Iteration 2

From iteration 1, we have a current merged metamodel. For short, this current metamodel will be called MOM (*Merged Organization Model*). In the second iteration, we *Match* and *Merge* TÆMS with MOM. For this, we take into account that the modeling constructs found in TÆMS are situated in the functional dimension.

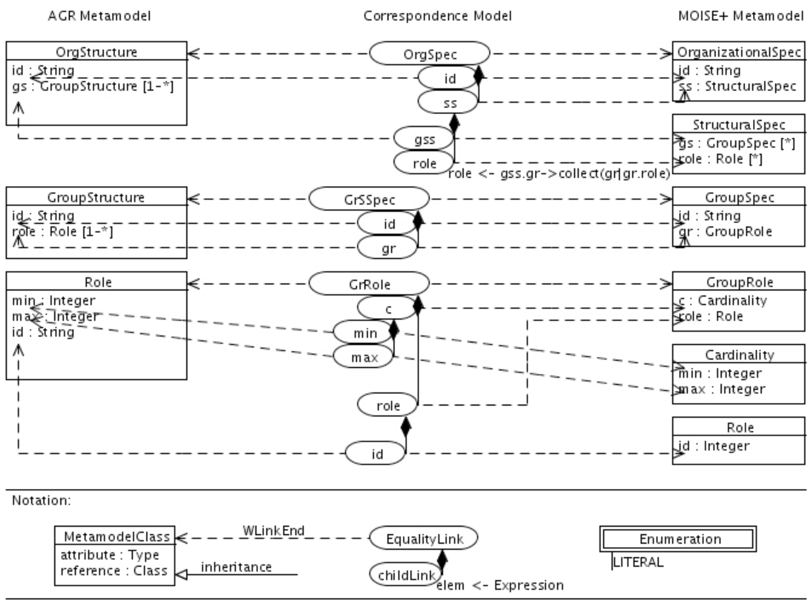


Fig. 1. *Match* of AGR and $\mathcal{M}$OISE+

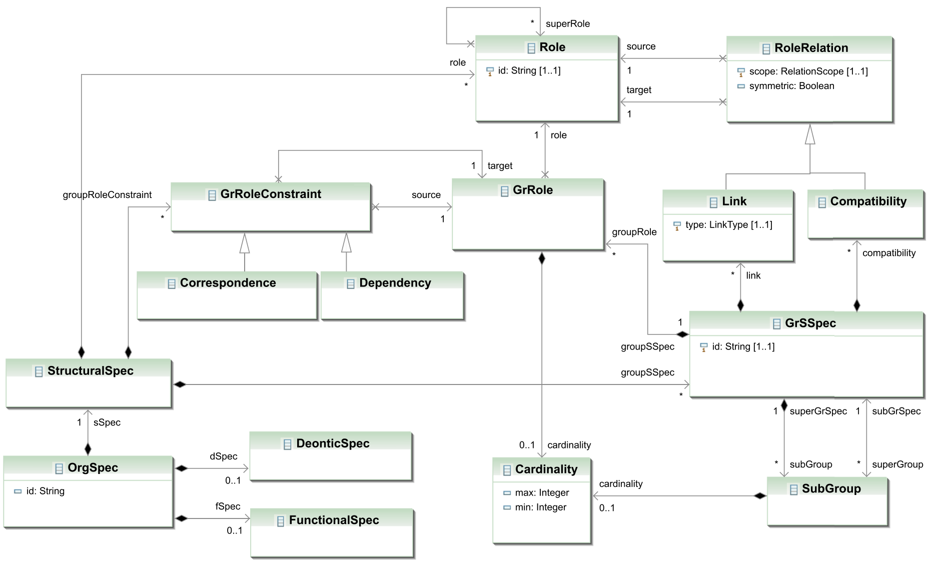


Fig. 2. Merge of AGR and $\mathcal{M}$OISE+

Match of TÆMS and previous MOM. In Fig. 3, we show the resulting mapping model from the matching of TÆMS with the previous AGR and $\mathcal{M}$OISE+ merge. Differently from AGR, the TÆMS metamodel focuses on task structures for agent groups. Comparing it with $\mathcal{M}$OISE+, we can say that it corresponds to elements from the functional dimension of $\mathcal{M}$OISE+.

Since `TAEMS!TaskStructure` is the root of a specification, it can be put in correspondence with the root element `MOM!OrgSpec`. Specifically, `TAEMS!TaskGroup` referred by `TAEMS!TaskStructure` contains equivalent information that is found on `MOM!SocialScheme`.

`TAEMS!TaskGroup` and `MOM!SocialScheme` are goal decomposition trees. Both have a root element and a set of elements hierarchically structured. In TÆMS, the basic element is called `AbstractTask` and is specialized in concrete `Task` and `Method`. `TAEMS!Task` points to sub-tasks and has an associated quality accumulation function (qaf) that tells how to combine sub-tasks to achieve a task. In MOM, the nodes are named `Goal`. `MOM!Goal` are combined into `MOM!Plan` that relates a head `MOM!Goal` with sub `MOM!Goal`. In this way, `TAEMS!AbstractTask` is mapped to `MOM!Goal` and `TAEMS! Task` to `MOM!Plan`. `TAEMS!Method` does not have a direct correspondent on MOM. Indirectly, via `TAEMS!AbstractGoal` it corresponds to a `MOM!Goal`. `TAEMS!Method` is similar to a `MOM!Goal` that is not head of any `MOM!Plan`.

TÆMS defines several quality accumulation functions (`TAEMS!QAF`). These are likely `MOM!PlanOperator` : they tell how to combine sub-goals to achieve some goal. So, `TAEMS!QAF` and `MOM!PlanOperator` can be put in correspondence. TÆMS defines `Q_SEQ_SUM` whose semantics is a superset of the `SEQUENCE` operator of MOM (sub-goal execution in sequence). They can be put in correspondence. And, the `Q_EXAC-TLY_ONE` of TÆMS conveys a similar meaning of `CHOICE` of MOM (only one sub-goal must be done to achieve the head goal). They also can be put in correspondence.

Merge of TÆMS and MOM. Given the previous *Match* and abiding by the GMRs (section 4.1), we achieve the second version of MOM as is shown in the Fig. 4. In the figure, beyond the common elements (`OrgSpec` , `GoalTree` , `AbstractGoal` , `CompositeGoal` and `QAF_OP`), there are elements unique to TÆMS and MOM. In MOM (from $\mathcal{M}$OISE+), there is `Mission`, a subset of `Goal` that has a `Cardinality` and is associated with `Role` (via normative relations not show in the figure). TÆMS defines `Resource`, an environmental element used to perform `Task`; `ResourceNLE` and `TaskNLE` are non-local effects relation between `Resource` and `Task` and between `Tasks`, respectively.

4.4 Other Iterations

We have done the same process integrating ISLANDER to the second MOM and OperA to the third MOM. Due to space limitations, we will briefly describe the result.

ISLANDER contributes to structural, interactive and normative modeling dimensions. Similar to AGR and $\mathcal{M}$OISE+ it has the concepts of roles and role relations. However, it does not have the concept of group specification. The focus of ISLANDER is in the interaction specification. In this regard, it adds to Fig. 4 a collection of new classes

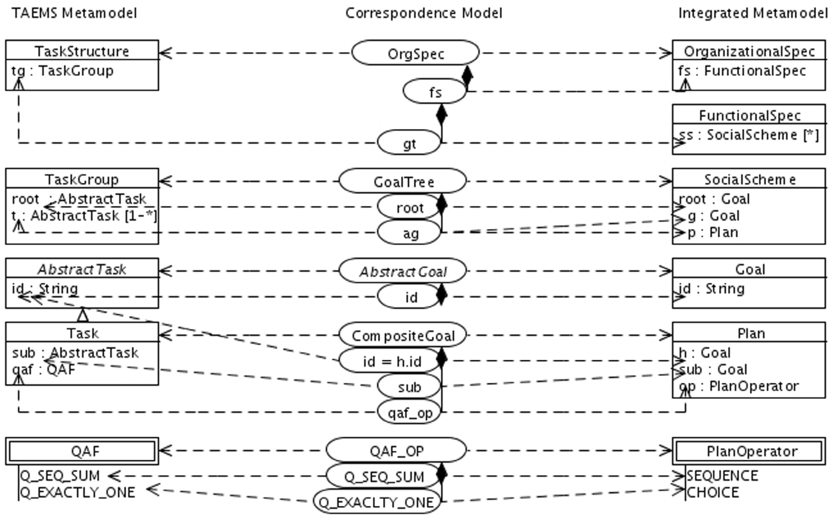


Fig. 3. *Match* of TÆMS and first *Merge*

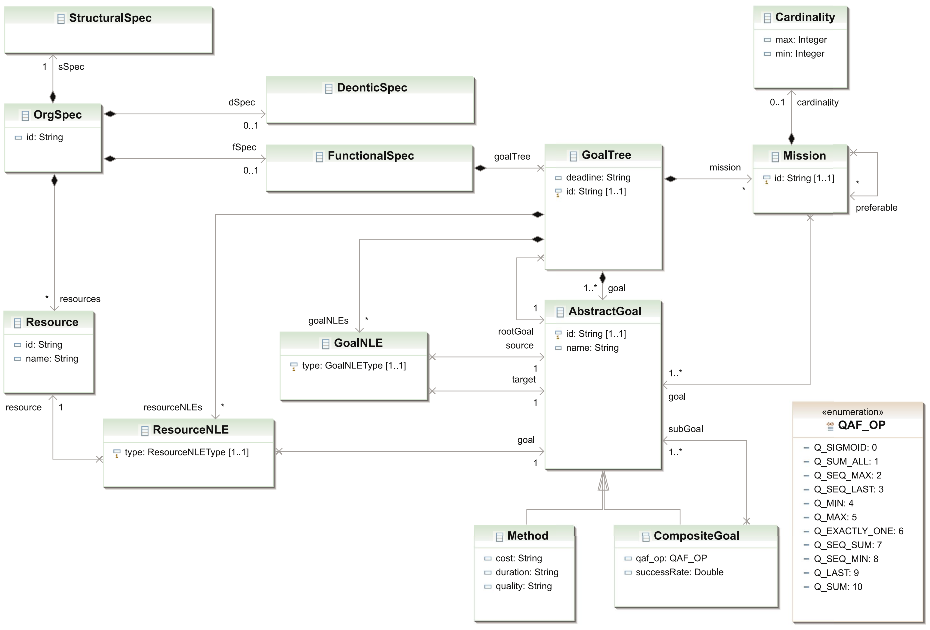


Fig. 4. *Merge* of TÆMS and first *Merge*

represented by the class `ISLANDER!PerformativeStructure`, which is placed as an immediate component of `MOM!OrgSpec`. Regarding norms, ISLANDER defines the class `Norm`.

OperA contributes to structural, dialogical, normative and functional dimensions. Its structural dimension is richer than ISLANDER but less developed than $\mathcal{M}$OISE+. Its interactive dimension is comparable to ISLANDER in its basic elements. The class `OPERA!InteractionStructure` is put in correspondence to `ISLANDER!PerformativeStructure`. The normative dimension is more developed than ISLANDER. Finally, it has the concept of objective, similar to goal, but nothing comparable to the functional view of TÆMS and $\mathcal{M}$OISE+.

5 Implementation and Application

The integration process described in the previous sections was implemented in a semi-automatic fashion using the Eclipse Modeling Framework[3] (EMF) and the Atlas Model Management Architecture[4] (AMMA). Not all organizational models have an explicit metamodel. The metamodels we used were re-engineered from XML schemes and formal definitions of the organizational models to be compatible with the EMF. To do the *Match* and *Merge* operation, we have used the AMW [22] plugin of AMMA. Regarding *TransfGen*, we have used the ATL [23] plugin of AMMA. ATL is a declarative rule-based transformation language.

In order to illustrate the application of our MOM in the context of organizational interoperability, we show in Fig. 5 part of the Guardian Angel scenario (section 2) specified in AGR and its corresponding transformation to $\mathcal{M}$OISE+, intermediated by the MOM. The specification in the left panel of the Fig. 5 describes in AGR the organization structure of the Health-Care System (HCS) depicted in Fig. 6. In HCS, Guardian Angel agents (GAs) can enter representing a patient. Once inside HCS, they can contact other agents representing care providers. The contact is intermediated by a broker agent. According to section 2, the GAs will interact with care-providers for several reason, one of them being to schedule appointments in behalf of a patient.

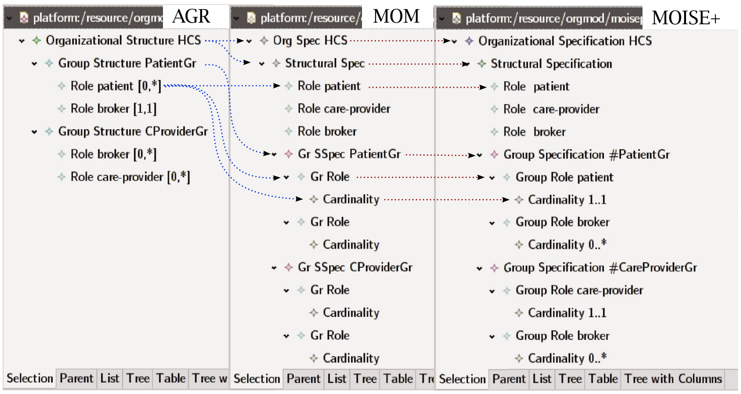
In this scenario, if we want to enable agents that interprets organizational specifications written in organizational models other than AGR, a possible solution is to translate from AGR to the other models. And this can be done by using the MOM as an intermediate format. For instance, if GA-2 in Fig. 6 is an agent that interprets $\mathcal{M}$OISE+, the transformation of Fig. 5 can be used by GA-2 to reason about the structure of HCS.

On the other hand, if GA-2 is an agent that interprets TÆMS there will be no possible translation from the organization structure of the HCS to a valid model in TÆMS. As we have noticed during the construction of MOM, AGR and TÆMS address orthogonal dimensions of agent organizations. Therefore, other mechanisms (like controllers of the agents inside an organizations) will be needed to make the agents interoperate.

Finally, if the GAs are built taking into account a comprehensive view of organizational models provided by an integration, then it can be made in a modular way to enable its participation in different organizational contexts.

[3] http://www.eclipse.org/modeling/emf/

[4] http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/



AGR

platform:/resource/orgmo

- Organizational Structure HCS
  - Group Structure PatientGr
    - Role patient [0,*]
    - Role broker [1,1]
  - Group Structure CProviderGr
    - Role broker [0,*]
    - Role care-provider [0,*]

Selection | Parent | List | Tree | Table | Tree w

MOM

platform:/resource/

- Org Spec HCS
  - Structural Spec
    - Role patient
    - Role care-provider
    - Role broker
    - Gr SSpec PatientGr
      - Gr Role
        - Cardinality
      - Gr Role
        - Cardinality
    - Gr SSpec CProviderGr
      - Gr Role
        - Cardinality
      - Gr Role
        - Cardinality

Selection | Parent | List | Tree | Table | Tr

MOISE+

platform:/resource/orgmod/moise

- Organizational Specification HCS
  - Structural Specification
    - Role patient
    - Role care-provider
    - Role broker
    - Group Specification #PatientGr
      - Group Role patient
        - Cardinality 1..1
      - Group Role broker
        - Cardinality 0..*
    - Group Specification #CareProviderGr
      - Group Role care-provider
        - Cardinality 1..1
      - Group Role broker
        - Cardinality 0..*

Selection | Parent | List | Tree | Table | Tree with Columns

Fig. 5. Transformation from AGR to $\mathcal{M}$OISE+

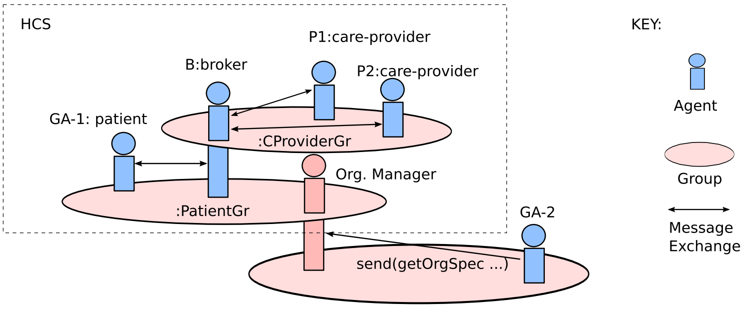


Fig. 6. Guardian Angel scenario

6 Related Work

In organization centered MASs [1], apart from communication languages [10], agent platforms and domain ontologies [11], the organizational model is another source of interoperability problems. In [24], the authors affirm: "Currently, in practice, agents are designed so as to be able to operate exclusively with a single given institution [organization], thus basically defying the open nature of the institution." In spite of this, we do not know about other work that address explicitly organizational interoperability problems.

In the area of methodologies for AOSE, the work [25] proposes an unified metamodel for interoperability between agent-oriented methodologies. To the extent that it deals with class-based metamodels of MASs and their integration, this work can be compared to ours. However, it has a different nature. While it deals with the interoperability among software engineers (and their tools) while designing MASs, we focus on interoperability among software agents during run-time w.r.t. organizational models.

With regard to model integration and interoperability in general, there is a vast literature. Complementing the references already indicated in the text, we cite [26] that proposes an architecture for model integration grounded on ontologies, [27] that surveys mappings in the domain of ontologies and [28] that presents the MOMENT framework as an alternative for the AMMA framework we have used to implement the integration of organizational models.

7 Conclusions and Future Work

In this paper we proposed an MDE approach for organizational models integration. Therefore, its contribution is twofold: the integration process itself and the use of MDE techniques in the open organized MAS engineering domain.

The integration process is an iterative and incremental process based on the application of the MDE operators *Match* and *Merge* to organization metamodels. In each step, an organization metamodel is matched and merged against a current integrated metamodel giving rise to a new integrated organization metamodel. Using the integration process, we have integrated the organization metamodels of AGR, $\mathcal{M}$OISE+, TÆMS, ISLANDER and OperA.

The integration result is a common metamodel that aims at providing interoperability among open and heterogeneous MAS. This can be achieved by combining the common metamodel with mapping models from/to each specific organization metamodel in order to synthesize runnable transformations that translates organizations' specifications between specific organizations' metamodels.

Considering these results, we intend to use the integrated organization metamodel as the central component for the definition of an architecture for organizational interoperability in the open organized MAS domain.

References

1. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
2. Lesser, V., et al.: Evolution of the GPGP/TAEMS domain-independent coordination framework. Autonomous Agents and Multi-Agent Systems 9, 87–143 (2004)
3. Hübner, J.F., Sichman, J.S., Boissier, O.: A model for the structural, functional, and deontic specification of organizations in multiagent systems. In: Bittencourt, G., Ramalho, G.L. (eds.) SBIA 2002. LNCS (LNAI), vol. 2507, pp. 118–128. Springer, Heidelberg (2002)
4. Esteva, M., Padget, J., Sierra, C.: Formalizing a language for institutions and norms. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, pp. 348–366. Springer, Heidelberg (2002)
5. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. PhD thesis, Utrecht University, SIKS Dissertation Series No. 2004-1 (2004)
6. Gutknecht, O., Ferber, J.: The MADKIT agent platform architecture. In: Wagner, T.A., Rana, O.F. (eds.) AA-WS 2000. LNCS, vol. 1887, pp. 48–55. Springer, Heidelberg (2001)
7. Hübner, J.F., Sichman, J.S., Boissier, O.: S-MOISE+: A middleware for developing organised multi-agent systems. In: Boissier, O., et al. (eds.) OOOP 2005: International Workshop on Organizations in Multi-Agent Systems, pp. 107–120 (2005)
8. Esteva, M., Rosell, B., Rodríguez-Aguilar, J.A., Arcos, J.L.: AMELI: an agent-based middleware for electronic institutions. In: AAMAS, pp. 236–243. IEEE Press, Los Alamitos (2004)

9. Coutinho, L., Sichman, J., Boissier, O.: Modelling dimensions for multi-agent system organizations. In: Multi-Agent Systems: Semantics and Dynamics of Organizational Models, ch. 2. IGI Global (to appear)
10. Labrou, Y., Finin, T., Peng, Y.: The interoperability problem: bringing together mobile agents and agent communication languages. In: HICSS 1999, Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences, vol. 8 (1999)
11. Erdur, R.C., Dikenelli, O., Seylan, I., Gürcan, Ö.: Semantically federating multi-agent organizations. In: Gleizes, M.-P., Omicini, A., Zambonelli, F. (eds.) ESAW 2004. LNCS, vol. 3451, pp. 74–89. Springer, Heidelberg (2005)
12. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: ACM SIGMOD International Conference on Management of Data, pp. 1–12 (2007)
13. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR 2003, First Biennial Conference on Innovative Data Systems Research, pp. 209–220 (2003)
14. Schmidt, D.C.: Model-driven engineering. IEEE Computer, 25–31 (February 2006)
15. Bézivin, J.: On the unification power of models. Soft. and Sys. Modeling 4, 171–188 (2005)
16. Szolovits, P., Doyle, J., Long, W.J., Kohane, P.S.G.: Guardian Angel: patient-centered health information systems. Technical Report 604, MIT, Lab. Computer Science (May 1994)
17. Davidsson, P.: Categories of artificial societies. In: Omicini, A., Petta, P., Tolksdorf, R. (eds.) ESAW 2001. LNCS (LNAI), vol. 2203, pp. 1–9. Springer, Heidelberg (2002)
18. Melnik, S., Bernstein, P.A., Halevy, A.Y., Rahm, E.: Supporting executable mappings in model management. In: ACM SIGMOD Conf. on Management of Data, pp. 167–178 (2005)
19. Fabro, M.D.D., Valduriez, P.: Semi-automatic model integration using matching transformations and weaving models. In: ACM Symposium on Applied Computing, pp. 963–970 (2007)
20. Pottinger, R., Bernstein, P.A.: Merging models based on given correspondences. In: VLDB 2003, Proc. of 29th Int. Conference on Very Large Data, pp. 826–873 (2003)
21. Lopes, D., Hammoudi, S., Abdelouahab, Z.: Schema matching in the context of model driven engineering: From theory to practice. In: Sobh, T., Elleithy, K. (eds.) Advances in Systems, Computing Sciences and Software Engineering, pp. 219–227. Springer, Heidelberg (2006)
22. Fabro, M.D.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: a generic model weaver. In: IDM 2005: 1ère Journée sur l'Ingénierie Dirigée par les Modèles (2005)
23. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
24. Dignum, F., et al.: Open agent systems??? In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 73–87. Springer, Heidelberg (2008)
25. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 62–77. Springer, Heidelberg (2005)
26. Kappel, G., et al.: Lifting metamodels to ontologies: a step to the semantic integration of modeling languages. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 528–542. Springer, Heidelberg (2006)
27. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. The Knowledge Engineering Review 18(1), 1–31 (2003)
28. Boronat, A., et al.: Formal model merging applied to class diagram integration. Electronic Notes in Theoretical Computer Science 166, 5–26 (2007)

MAS Modeling Based on Organizations

Estefanía Argente, Vicente Julian, and Vicent Botti[*]

Departamento de Sistemas Informaticos y Computacion,
Universidad Politecnica de Valencia,
C/Camino de Vera s/n, 46022, Valencia, Spain
{eargente,vinglada,vbotti}@dsic.upv.es

Abstract. An agent organization model is proposed based on four main concepts: organizational unit, service, environment and norm. These concepts are integrated in ANEMONA meta-models, which are extended in order to include all entities needed for describing the structure, functionality, dynamic, normalization and environment of an organization.

Keywords: Meta-model, Multi-agente systems, Organization.

1 Introduction

Organizational models have been recently used in agent theory to model coordination in open systems and to ensure social order in MAS [1,2]. Agent Organizations rely on the notion of openness and heterogeneity and include the integration of organizational and individual perspectives and the dynamic adaptation of models to organizational and environmental changes [3].

Meta-modeling is a mechanism that allows defining modeling languages in a formal way, establishing the primitives and syntactic-semantic properties of a model [4]. For example, INGENIAS [5] and ANEMONA[6] methods offer several meta-models for MAS analysis and design, by means of their component description (organizations, agents, roles), functionality (goals and tasks), environment (resources and applications), interactions and agent internal features, such as autonomy and mental state processing. INGENIAS follows an iterative development process based on *Rational Unified Process* (RUP). It is supported by powerful tools for modeling, design and code generation. ANEMONA, based on INGENIAS, is a MAS methodology for developing Holonic Manufacturing Systems. They both employ UML notation language for meta-model descriptions. However, they lack of a specific normative description, a deeper analysis of the system dynamics and an open system perspective.

Other MAS frameworks, such as MOISE[7] or E-Institutions[1], do specially focus on the normative specification of the system, but do not take into account the environment description or a more detailed analysis of the organization structure and functionality.

[*] This work has been partially funded by TIN2005-03395 and TIN2006-14630-C03-01 projects of the Spanish goverment, FEDER funds and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

In this work, an integration of several methods and modeling languages (such as ANEMONA, AML[8], MOISE, E-Institutions) is proposed for describing the main features of an organization: its structure, functionality, dynamics, environment and norms. In this way, four main concepts are employed: organizational unit, service, norm and environment. These concepts have been extracted from human organizational approaches [9,10,11] and also from multiagent system works [12,8] and service oriented architectures[1]. They are used to represent: (i) how entities are grouped and connected between them and their environment; (ii) which functionality they offer, and which services are employed to manage dynamic entry/exit of agents in the organization; and (iii) which restrictions are needed for controlling entity behavior inside the system.

The proposed MAS modeling employs six different meta-models, which extend the ANEMONA ones: the *organization meta-model*, that describes system entities (agents, organizational units, roles, norms, resources, applications) and how they are related to each other (i.e. social relationships, functionality needed or offered); the *activity meta-model*, that details the specific functionality of the system (services, tasks and goals); the *interaction meta-model*, that defines system interactions, activated by means of goals or service usage; the *environment meta-model*, that describes system applications and resources, agent perceptions and effects and also service invocation through its ports; the *agent meta-model*, that describes concrete agents; and finally the *normative meta-model*, that details organizational norms that agents must follow.

A case-study example based on the travel domain is used to provide a better comprehension of the meta-models. Hotel chains and flight companies offer information about their products (hotels, flights), booking facilities and advance payment. Their functionality is defined using services and it is controlled with norms that describe, for example, which are the minimum services that providers must register in the system in order to participate inside; how services are described (service profiles and processes); or in which order services must be served.

In this paper, the main extensions to ANEMONA meta-models are related, using UML notation language, following GOPPR [13] restrictions. All relationships have a specific prefix that indicates: **O** for organization; **GT** for goals and tasks; **WF** for work flow; **AGO** for social relations; **E** for environment; **N** for norms and **I** for interactions. A *Role* primitive is employed to establish the direction of the relationship, having as suffix an *O* for origin and *D* for destiny. All meta-model extensions are graphically emphasized in dark color. Due to lack of space, only those meta-models with more extensions are explained. More specifically, a description of the *organization meta-model* is detailed in section 2; how services are described using the *activity meta-model* is explained in section 3; extensions to the *environment meta-model* are shown in section 4; whereas section 5 describes how rules are modeled using the *normative meta-model*. Finally, conclusions and discussion are detailed in section 6.

[1] http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

2 Modeling MAS Organizations

An agent organization is defined as a social entity composed of a specific number of members that accomplish several distinct tasks or functions and are structured following some specific topology and communication interrelationship in order to achieve the main aim of the organization [14]. Agent organizations assume the existence of global goals, outside the objectives of any individual agent, and they exist independently of agents [3].

ANEMONA meta-models offer the *Abstract Agent* (A-Agent) notion [6], which allows defining agent collections as unique entities of a high-level description, modeled as virtual single agents. But A-Agents can be later refined and specified internally, defining all their components (simple agents or groups of agents). Thus, an A-Agent is defined in a recursive way, being an atomic entity or a multi-agent system (with unique entity) composed of A-Agents not necessarily equal.

In this paper, this A-Agent entity has been extended with the **Organizational Unit** (OU) concept, that describes the existing groups of members of the organization. These units have a specific internal structure. They also define several roles or positions that describe a set of functionalities (services offered and required) and goals that represent the organizational expectation for each position. OUs also include resources and applications, that can be accessed by specific members of the organization. And finally, they include all norms that control their members' behavior.

The proposed *organization meta-model* integrates this *Organizational Unit* concept and contains four views: structural, functional, social and dynamic. The first three ones are extensions of those ones employed in ANEMONA, whereas the new dynamic view is used to specify which are the services that an OU must offer to control and manage entry and exit of its entities.

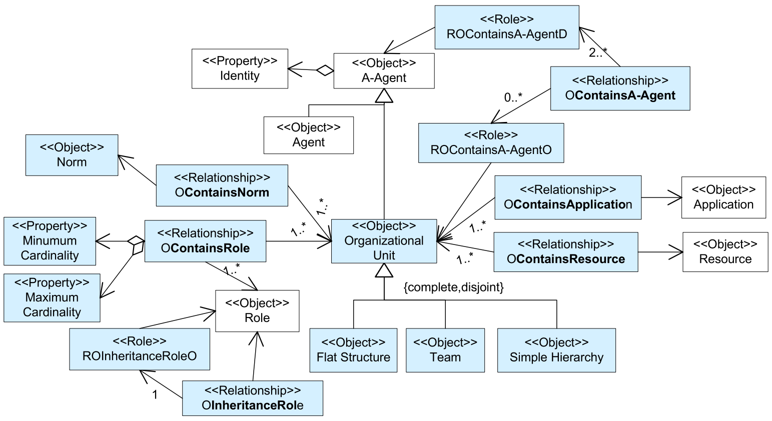

<<Role>> ROContainsA-AgentD
<<Property>> Identity
<<Object>> A-Agent
2..*
<<Relationship>> OContainsA-Agent
0..*
<<Object>> Agent
<<Role>> ROContainsA-AgentO
<<Object>> Norm
<<Relationship>> OContainsNorm
1..*
<<Relationship>> OContainsApplication
<<Object>> Application
<<Property>> Minumum Cardinality
<<Relationship>> OContainsRole
1..*
<<Object>> Organizational Unit
1..*
<<Relationship>> OContainsResource
<<Object>> Resource
1..*
1..*
<<Property>> Maximum Cardinality
<<Object>> Role
{complete,disjoint}
<<Role>> ROInheritanceRoleO
<<Object>> Flat Structure
<<Object>> Team
<<Object>> Simple Hierarchy
1
<<Relationship>> OInheritanceRole

Fig. 1. Organization Meta-model. Structural view.

The **structural view** defines which are the “static” components of the organization, i.e. all elements that are independent of the final executing entities (figure 1). Thus, the system is composed of *Organizational Units* (OU), that can also include other units in a recursive way. Internally, their members are arranged in a hierarchy, team or plain structure. The composition of these units facilitates designing more complex and elaborated structures, such as matrix, federation, coalitions or congregations [14]. The OU acts as a group of agents (*OContainsA-Agent*), but also as their environment. Hence, it contains both resources (*OContainsResource*) and applications (*OContainsApplication*) that can be used by the OUs entities. It also defines the allowed roles inside the unit (*OContainsRole*) and all norms that control their behavior (*OContainsNorm*).

In the travel case study (figure 3.A), the *TravelAgency* organizational unit represents the whole travel system. The *Client* role represents the final user that asks for information on hotels or flights, orders booking rooms or flight seats and even might pay in advance. The *Provider* role offers searching and booking service functionality. Finally, the *Payee* role is responsible for controlling the advance payment. As descriptions and functionalities for travel search and booking services might be rather different for hotels and flights, two organizational units (*FlightUnit* and *HotelUnit*) have been defined, focused on their specific products. In these units, both client and provider roles are specialized into more specific roles (ex. *FlightClient* and *FlightProvider*).

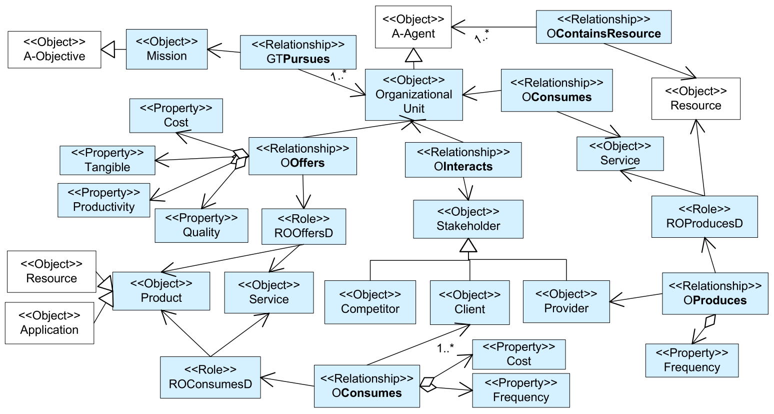


Fig. 2. Organization Meta-model. Functional view. Mission.

The **functional view** describes the organizational mission and how each organizational unit behaves, both externally and internally. It contains three subviews: mission, external functionality and internal functionality.

The *mission* (figure 2) defines organizational global goals (*GTPursues*), who are the stakeholders that interact with the organization (*OInteracts*), which are the results of the organization (*OOffers* products or services), how these results are consumed by its clients (*OConsumes*) and what the organization needs from

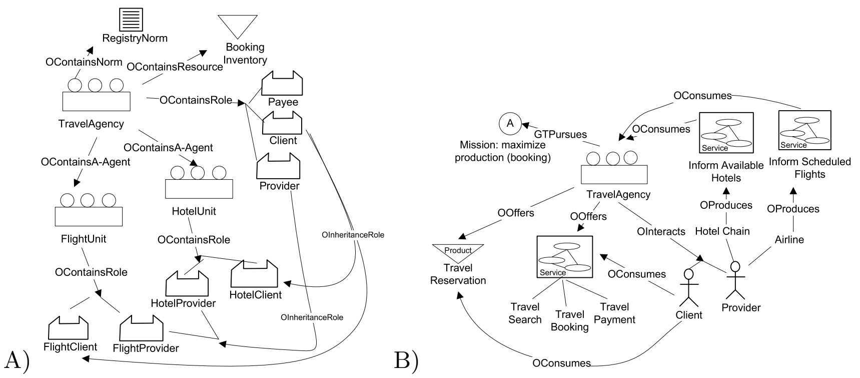


Fig. 3. Example of the organizational model diagram for the travel agency case-study: A) structural view; B) functional view (mission)

its providers (*OProduces* services or resources, *OConsumes* services, *OContainsResource*). In the case-study example (figure 3.B) the system (*TravelAgency* unit) offers the travel reservation product, consumed by its clients (tourists or businessmen). It also offers several services for travel searching, booking and payment. Moreover, this system requires that some providers (hotel chains and airlines) supply all needed information about hotels and flights.

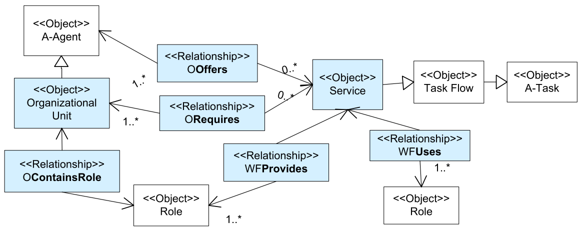


Fig. 4. Organization Meta-model. Functional view. External functionality.

The *external functionality* of an A-Agent (figure 4) represents the set of services that this entity offers to other A-Agents (*OOffers* relationship), independently of the final agent that makes use of them. Moreover, a set of services required by OUs can also be defined. These services represent all functionality that needs to be "hired" to other A-Agents. The *ORequires* relationship is similar to "job offer advertising" of human organizations, in the sense that it represents a necessity of finding agents capable of providing these required services as members of the unit. All features, abilities and permissions of providers and clients of these services are modeled by means of roles, using *WFProvides* and *WFUses* relationships. The *OOffers* relationship of this subview is the same one of the *mission* subview, but offered services are more specified. The *ORequires* relation is also related with the *OConsumes* relation of the *mission* subview. In this case, *ORequires* is connected to services that must be provided inside the OU, whereas the *OConsumes* relationship is related to services that are needed by the OU but it is not yet defined whether they are executed inside or outside the organization (i.e. invoke to external entities).

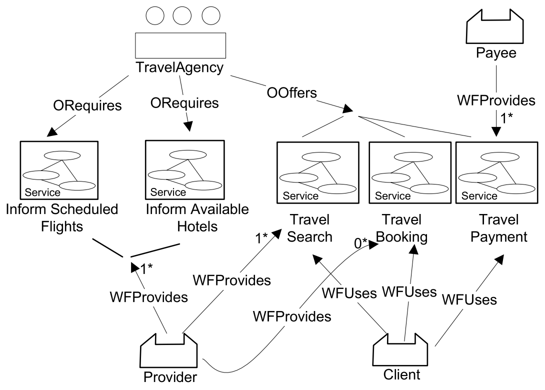


Fig. 5. Example of the Organizational model diagram (external functionality) for the travel case-study

In the travel case-study example (figure 5), the *TravelAgency* unit offers *TravelSearch*, *TravelBooking* and *TravelPayment* services to agents playing the *client* role. Moreover, *provider* agents must supply at least an information service, invoked in the *TravelSearch*. Thus, any agent willing to play a provider role has to be capable of providing a service of this kind. However, the *TravelBooking* service is not compulsory, so providers can freely decide whether to offer it or not. The *TravelPayment* service is assigned to the *Payee* role.

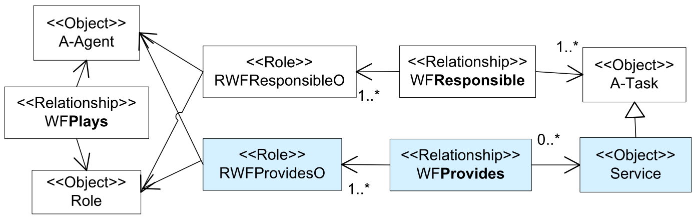


Fig. 6. Organization Meta-model. Functional view. Internal functionality.

Finally, the *internal functionality* of an A-Agent (figure 6) is defined by its tasks (*WFResponsible*), which are delimited by the roles that the entity plays (*WFPlays*) and the services provided by these roles (*WFProvides*). For example, the *Bank* agent (figure 8.A) plays the *Payee* role in the travel case-study, implementing the *TravelPayment* service functionality.

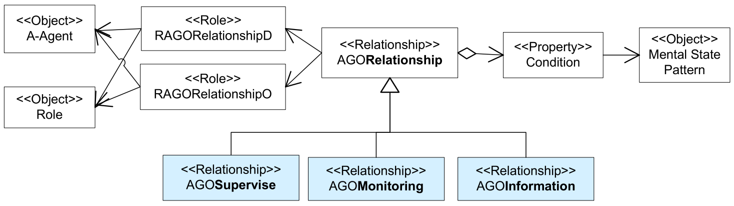

<<Object>> A-Agent
<<Object>> Role
<<Role>> RAGORelationshipD
<<Role>> RAGORelationshipO
<<Relationship>> AGORelationship
<<Property>> Condition
<<Object>> Mental State Pattern
<<Relationship>> AGOSupervise
<<Relationship>> AGOMonitoring
<<Relationship>> AGOInformation

Fig. 7. Organization Meta-model. Social view.

The **social view** (figure 7) describes roles and A-Agent social relationships, divided into three types: supervision, monitoring and information. This social view integrates [15] and [7] works in this meta-model approach.

The *AGOInformation* relationship describes how information or knowledge links are established inside the organization. If two A-Agents are connected by this type of link, then they are entitled to know each other and communicate relevant information. The *AGOMonitoring* relationship implies a monitoring process of agent activity, so the monitor agent is responsible for controlling tasks of its monitored agents. Finally, the *AGOSupervise* relationship implies that a (supervisor) agent transfers or delegates one or more goals to its subordinate agent, which is obliged to include these objectives as its own and pursue them.

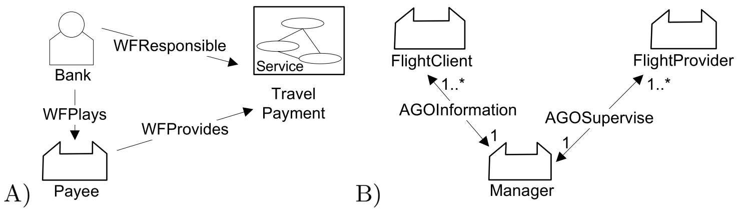


Fig. 8. Example of the Organizational Model diagram for the travel agency case-study: A) internal (functional) view. A *Bank* agents plays the *Payee* role; B) social view.

In the travel case-study example (figure 8.B), the *FlightUnit* has been modeled using a hierarchical structure in which there is a supervisor (*manager* role) that receives all flight requests from clients and invokes *FlightProvider* services, also controlling their behavior.

The **dynamic view** (figure 9) defines the pattern designs for organizational unit services, that enable managing all its structural and dynamic components. These services are divided into structural, informative and dynamic services. The *structural services* are focused on adding or deleting norms, roles or organizational units. The *informative services* provide information about the structure of the organization. And the *dynamic services* manage the inclusion and exit of agents into the unit and the role adoption. These last services need to be published in an open system for allowing external agents to participate inside.

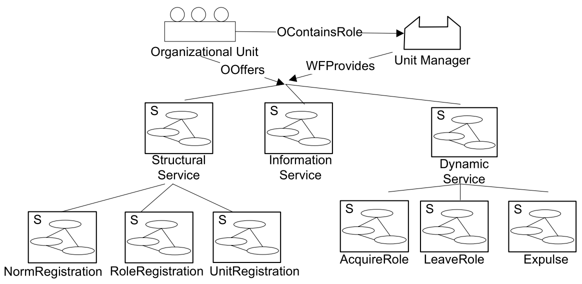


Fig. 9. Pattern Design for the dynamic view (Organization Model)

3 Modeling MAS Services

Services represent a functionality that agents offer to other entities, independently of the concrete agent that makes use of it. Its main features are: (i) synchronization, that implies interaction between entities that offer the service and those ones that require and use it; (ii) publishing, so the service is registered in a service directory and other entities can find it; (iii) participation, i.e. entities that consume the service can differ through time; (iv) entity standardization, as service consumers and providers are related to specific roles, for which restrictions are defined through norms, resource access permissions, etc.; (v) functionality standardization, as services are described in terms of inputs, outputs, preconditions and postconditions, making easier the description its functionality; (vi) tangibility, as services usually produce tangible products which can be employed for evaluating both quality, service efficiency and client satisfaction; and (vii) cost, i.e. service production and consumption imply some costs and/or benefits.

In the **activity meta-model** (figure 10), service system functionality is described by means of its profile and A-Tasks in which a service is split (*WFSplits* relationship). The *ServiceProfile* concept describes activation conditions of the service (preconditions), its input, output parameters and its effects over the environment (postconditions). It can be lately used in an OWL-S service description. The *A-Task* concept (figure 11.A) describes the service functionality. It represents both concrete tasks, task-flows or service composition (*WFInvokes* relationship). A task-flow description (figure 11.B) relates tasks with their environment: usage of resources and mental entities (*WFConsumes*, *WFProduces*), usage of applications (*WFEmploys*), task sequence order (*WFConnects*, *WFInvokes*), task composition (*WFSplits*) and task assignment to agents (*WFResponsible*) and its execution (*WFExecutes*).

The activity model diagram for *TravelSearch* service of the case-study example is shown in figure 12. This service is described using the "Travel Searching" profile and contains four tasks: *CheckPlace*, that checks inputs (country and city); *FlightSearch* and *HotelSearch* (concurrent tasks that invoke *InformScheduledFlights* and *InformAvailableHotels* services, respectively); and *TravelFilter*, that selects the best hotels and flights. The task flow description for the *TravelSearch* service is shown in figure 12.B.

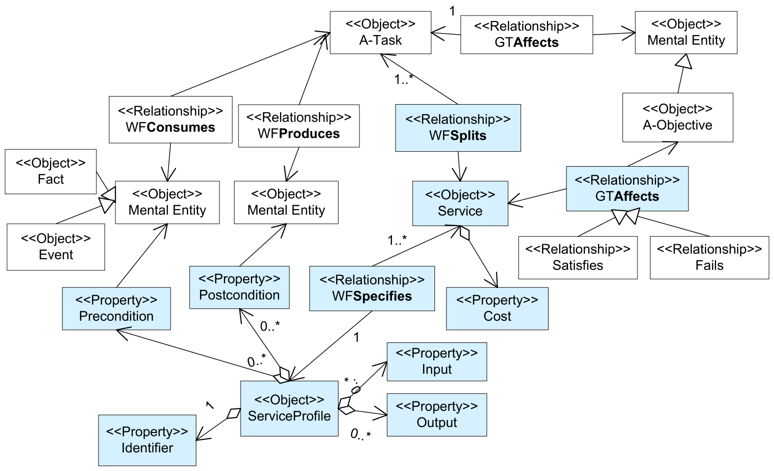

<<Object>> A-Task
<<Relationship>> GTAffects
<<Object>> Mental Entity
<<Object>> A-Objective
<<Relationship>> WFConsumes
<<Relationship>> WFProduces
<<Relationship>> WFSplits
<<Object>> Fact
<<Object>> Event
<<Object>> Mental Entity
<<Object>> Service
<<Relationship>> GTAffects
<<Relationship>> Satisfies
<<Relationship>> Fails
<<Property>> Precondition
<<Property>> Postcondition
<<Relationship>> WFSpecifies
<<Property>> Cost
<<Property>> Input
<<Property>> Output
<<Object>> ServiceProfile
<<Property>> Identifier
1
1..*
0..*
*

Fig. 10. Activity Meta-model. Service description.

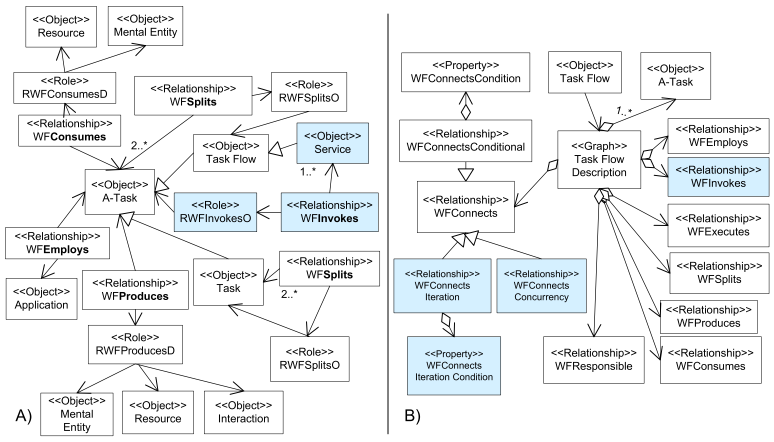
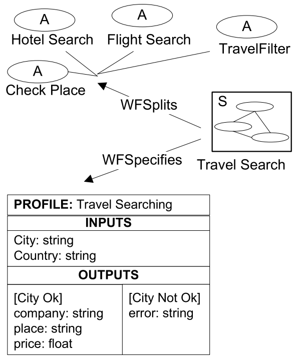

A)
<<Object>> Resource
<<Object>> Mental Entity
<<Role>> RWFConsumesD
<<Relationship>> WFConsumes
<<Relationship>> WFSplits
<<Role>> RWFSplitsO
<<Object>> Task Flow
<<Object>> Service
<<Object>> A-Task
<<Role>> RWFInvokesO
<<Relationship>> WFInvokes
<<Relationship>> WFEmploys
<<Object>> Application
<<Relationship>> WFProduces
<<Object>> Task
<<Relationship>> WFSplits
<<Role>> RWFProducesD
<<Role>> RWFSplitsO
<<Object>> Mental Entity
<<Object>> Resource
<<Object>> Interaction
B)
<<Property>> WFConnectsCondition
<<Object>> Task Flow
<<Object>> A-Task
<<Relationship>> WFConnectsConditional
<<Graph>> Task Flow Description
<<Relationship>> WFEmploys
<<Relationship>> WFInvokes
<<Relationship>> WFConnects
<<Relationship>> WFExecutes
<<Relationship>> WFSplits
<<Relationship>> WFConnects Iteration
<<Relationship>> WFConnects Concurrency
<<Relationship>> WFProduces
<<Property>> WFConnects Iteration Condition
<<Relationship>> WFResponsible
<<Relationship>> WFConsumes
2..*
1..*

Fig. 11. Activity Meta-model. A) Task description; B) Task Flow Description.

4 Modeling MAS Environment

Based on human organizations [16,11], environment should be modeled with two different perspectives: structural and functional. The structural perspective describes which are the components of the system (agents, objects, resources), how they are related (i.e. agent groups, behavioral norms, resource access) and how these elements are conceptually represented, by means of an ontology. The functional perspective determines which are the activities related with the environment, i.e., how agent communication is produced (direct or indirect messages, using specific environment elements, etc.), how agents can perceive and act over

A)


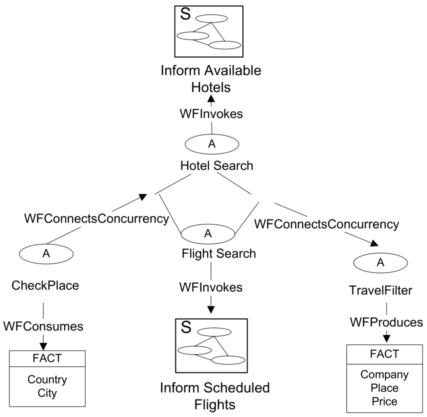
B)


Fig. 12. Activity model diagram for the travel case-study example: A) *TravelSearch* service description; B) *TravelSearch* service task-flow

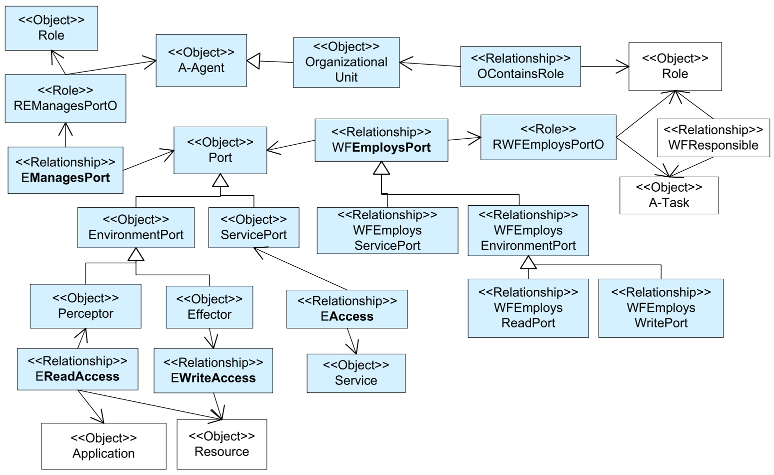
the environment and how agents are connected to other types of entities such as objects, applications or resources.

The proposed *environment meta-model* (figure 13) focuses on the description of its elements (resources, applications and mental entities), perceptions and actions over the environment, and permission accesses for using these elements.

Resources represent environment objects that do not provide a specific functionality, but are indispensable for task execution [5]. They can be consumable or not, have an initial state, a lower and upper threshold and a capacity granularity. As regards applications, they represent functional interfaces that are described with a name, several parameters, preconditions, postconditions and results.

Agent perceptions and actions are described using the *EnvironmentPort* concept, which is a specialization of the *Port* entity. This concept has been extracted from AML language modeling [8], in which a port represents an interaction point between an entity and other model elements. Two kinds of ports have been defined: environment and service ports. The environment port allows lecture and/or write access to resources or applications. The *Perceptor* port establishes how agents can obtain information from resources and applications. The *Effector* port allows agents to modify resource data. The *EManagesPort* relationship indicates who manages and controls the environment port access. The *WFEmploysPort* relationship represents which roles are allowed to use the port and in which way (*WFEmploysReadPort* for obtaining information, *WFEmploysWritePort* for creating or modifying environment information).

The *ServicePort* concept represents the publishing feature of the service, i.e. the contact point or grounding mechanism for service access. The entity in charge of publishing it (in a service directory, for example) is represented with the *EManagesPort* relationship.



<<Object>> Role
<<Object>> A-Agent
<<Object>> Organizational Unit
<<Relationship>> OContainsRole
<<Object>> Role
<<Role>> REManagesPortO
<<Object>> Port
<<Relationship>> WFEmploysPort
<<Role>> RWFEmploysPortO
<<Relationship>> WFResponsible
<<Relationship>> EManagesPort
<<Object>> A-Task
<<Object>> EnvironmentPort
<<Object>> ServicePort
<<Relationship>> WFEmploys ServicePort
<<Relationship>> WFEmploys EnvironmentPort
<<Object>> Perceptor
<<Object>> Effector
<<Relationship>> EAccess
<<Relationship>> WFEmploys ReadPort
<<Relationship>> WFEmploys WritePort
<<Relationship>> EReadAccess
<<Relationship>> EWriteAccess
<<Object>> Service
<<Object>> Application
<<Object>> Resource

Fig. 13. Environment meta-model. Port access.

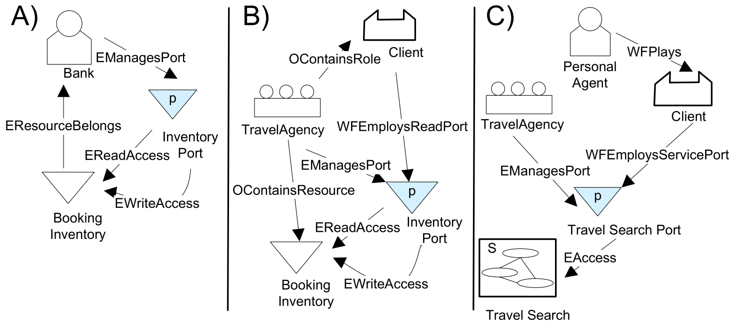


Fig. 14. Case-study example: **A)** the *bank* agent contains the *Booking Inventory* and manages its access; **(B)**The *TravelAgency* unit contains the *Booking Inventory* and manages its access; **(C)** The *TravelAgency* unit publishes *TravelSearch* service, which is used by agents playing the *client* role

For the travel case-study, an example of a resource belonging to a specific agent is shown in figure 14.A, in which the *Bank* agent controls access to the *Booking Inventory* by means of the *Inventory Port*. However, in many problems the resource does not belong to a specific agent, but to the environment of a group of agents. In this case, the organizational unit that represents this group contains this resource and manages its access through a resource port. For example, in figure 14.B, the *TravelAgency* unit contains the *Booking Inventory* resource, which can be read or modified, but the *client* role defined in this unit is only empowered to read access. Finally, an example of a service port access is shown in figure 14.C, in which a *PersonalAgent* playing the *client* role is allowed to make use of the *TravelSearch* service. The *TravelAgency* unit is in charge of publishing this service (represented by the *EManagesPort* relationship).

5 Modeling MAS Norms

Norms have been widely used as mechanisms to limit human autonomy inside societies, in order to solve problems of coordination, specially when total and direct social control cannot be exerted. In open multi-agent systems, norms have been considered as a key issue for managing the heterogeneity, autonomy and diversity of interests of agents [17].

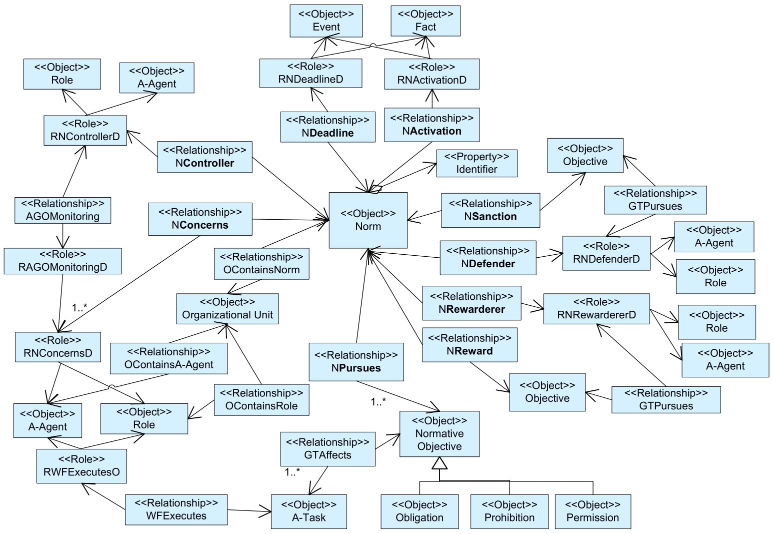


Fig. 15. Normative meta-model

The proposed **normative meta-model** (figure 15) describes the **Norm** concept, which represents a specific regulation, expressed by means of a *Normative Objective* (Obligation, Permission or Prohibition). This regulation affects or concerns A-Agents or Roles (*NConcerns* relationship), whose actions are controlled by the normative objective (*WFExecutes* and *GTAffects* relationships). The norm also indicates who is in charge of monitoring that the norm is satisfied (*NController* relationship) and who is responsible for punishments (*NSanction* and *NDefender* relationships) and/or rewards (*NReward* and *NRewarderer* relationships). Finally, *NActivation* relationship specifies all facts and events of the environment that provoke the activation of the norm. Its deactivation (*NDeadline*) is produced when the normative objective or the deadline is satisfied.

In figure 16, the pattern design for an obligation norm is shown. A sanction is created when the deadline has been reached and compulsory tasks have not been satisfactorily executed yet.

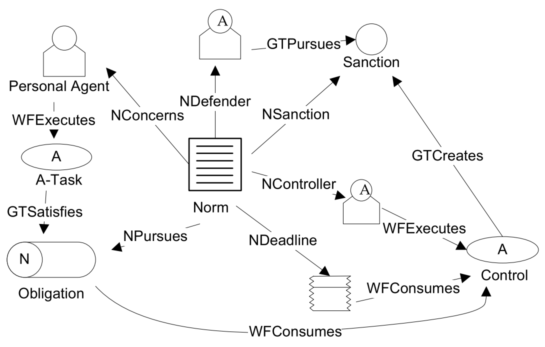


Fig. 16. Pattern design of an obligation

6 Discussion

An extension of ANEMONA meta-models has been proposed, in order to include concepts of organizational unit, service and norm. These concepts have been extracted from human organizational approaches, from multiagent systems works and from service oriented architectures, being integrated in a framework for modeling organizations. In this way, the main features of an organization can be described: its structure, functionality, dynamics, environment and norms. Thus, the *organization meta-model* describes its components, relationships and connections with its environment. The *activity meta-model* details offered and required services, their tasks and objectives. The *environment meta-model* captures system resources and applications, agent perceptions and effects and port accesses permissions. Moreover, organization rules are expressed with the *normative meta-model*. Finally, the *agent meta-model* details concrete responsibilities of agents and their internal functions, and the *interaction meta-model* defines specific interactions between agents and service invocation (using service ports). These two last meta-models have not been included in this paper due to lack of space, but they are mainly the same as in ANEMONA.

Regarding related work on MAS organizational modelling, there are different interesting approaches, standing out AGR [18], MOISE [7], ODML [20], AML [8], OMNI [21] and OMACS [23]. The AGR model [18] is based on agent, groups and role concepts. It was lately extended in the AGRE[19] work (E for environment).

MOISE^{Inst} model [7] includes structural, functional and deontic views. Its structural view is related with our organization meta-model, detailing roles, groups and relationships. In our proposal, the environment is also modelled and the internal topology of groups is considered as well. Its functional view describes plans and missions to achieve goals, similarly to our A-Objectives. In our approach, services required and offered are modelled too, and agent interactions are deeply described in the interaction meta-model. Finally, MOISE^{Inst} deontic view describes permissions and obligations of roles, including sanctions. Our normative meta-model also incorporates rewards.

ODML [20] uses a basic underlying model of organizations for performance prediction of the multiagent organization. Their existing organizational models [12] have served as a basis for our topological analysis [14]. AML [8] extends UML with agent concepts, including resources, environment, organizational units and services, but it lacks of a normative modeling. Our proposal has adopted AML environment perspective, using ports for accessing services and resources. Moreover, our meta-models are integrated in an iterative process, such as in INGENIAS or ANEMONA methodologies.

OMNI [21] offers Normative, Organizational and Ontological Dimensions. It makes use of ISLANDER [1] and AMELI [22] framework for MAS implementation. The mission of the organization, its norms and rules, roles, groups and concrete ontological concepts are detailed. It is also based on contracts, used for acquiring roles and controlling agent interactions. In our proposal, these contract specifications can be employed to better define the organizational services in the dynamic view of the organization meta-model.

Finally, OMACS [23] defines a meta-model for MAS that allows the system to design its own organization at runtime. It is based on agent capabilities (similar to our agent meta-model, in which tasks and services that an agent is responsible for are defined), role assignments (described in our organization meta-model) and policies, which include behavioral and reorganization policies (defined in our normative meta-model) and assignment policies (described in our organization and environment meta-models using access restrictions on resources and services).

The Organizational MAS modeling approach presented in this paper has been integrated in an iterative process of system development, in which several methodological guidelines are employed for describing the mission of the organization, its productive tasks and processes, its organizational dimensions and topological structure, its decision and information processes, its dynamics and normative behavior and its reward system. Moreover, a BNF language for describing norms has been developed. It allows defining restrictions on service usage, registration and provision. Furthermore, a graphical development tool is currently being implemented, that helps designers with diagram model construction and automatic code generation.

References

1. Esteva, M., de la Cruz, D., Sierra, C.: ISLANDER: an Electronic Institution Editor. In: Proc. AAMAS 2002, pp. 1045–1052 (2002)
2. Dignum, V., Meyer, J., Wiegand, H., Dignum, F.: An organization-oriented model for agent societies. In: Proc. RASTA 2002 (2002)
3. Dignum, V., Dignum, F.: A landscape of agent systems for the real world. Tech. Report Inst. Information and Computer Sciences, Utrecht. Univ. (2006)
4. van Gigch, J.P.: System Design Modeling and Metamodeling. Plenum Press, New York (1991)
5. Gomez, J., Fuentes, R., Pavon, J.: The INGENIAS Methodology and Tools. In: Agent-oriented Methodologies, pp. 236–276. Idea Publishing Group, USA (2005)
6. Botti, V., Giret, A.: ANEMONA: A multi-agent methodology for Holonic Manufacturing Systems. Series in Advanced Manufacturing. Springer, Heidelberg (2008)

7. Gateau, B., Boissier, O., Khadraoui, D., Dubois, E.: Moise-inst: An organizational model for specifying rights and duties of autonomous agents. In: Proc. CoORG (2005)
8. Cervenka, R., Trencansky, I.: AML. The Agent Modelling Language. Whitestein Series in Soft. Agent Tech. and Autonomic Computing. Birkhäuser, Basel (2007)
9. Robbins, S.: Organizational Behavior. Pearson Prentice Hall (2007)
10. Moreno-Luzon, M., Peris, F., Gonzalez, T.: Gestión de la Calidad y Diseño de Organizaciones. Prentice Hall, Pearson Education (2001)
11. Hodge, B.J., Anthony, W., Gales, L.: Organization Theory: A Strategic Approach. Prentice Hall, Englewood Cliffs (2002)
12. Horling, B., Lesser, V.: A survey of multiagent organizational paradigms. The Knowledge Engineering Review 19, 281–316 (2004)
13. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment. In: Constantopoulos, P., Vassiliou, Y., Mylopoulos, J. (eds.) CAiSE 1996. LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
14. Argente, E., Palanca, J., Aranda, G., Julian, V., Botti, V., Garcia-Fornes, A., Espinosa, A.: Supporting Agent Organizations. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS, vol. 4696, pp. 236–245. Springer, Heidelberg (2007)
15. Grossi, D., Dignum, F., Dastani, M., Royakkers, L.: Fundations of organizational structures in multiagent systems. In: Proc. AAMAS 2005, pp. 690–697 (2005)
16. Weyns, D., Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments for multiagent systems. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS (LNAI), vol. 3374, pp. 1–47. Springer, Heidelberg (2005)
17. López, F., Luck, M., d'Inverno, M.: A normative framework for agent-based systems. Computational and Mathematical Organization Theory 12, 227–250 (2006)
18. Ferber, J., Gutkenecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
19. Ferber, J., Michel, F., Baez, J.: AGRE: Integrating Environments with Organizations. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS, vol. 3374, pp. 48–56. Springer, Heidelberg (2005)
20. Horling, B., Lesser, V.: Using ODML to model multi-agent organizations. In: Proc. IEEE/WIN/ACM INt. Conf. on Intelligent Agent Technology, pp. 72–80 (2005)
21. Dignum, V., Vazquez, J., Dignum, F.: OMNI: Introd. social structure, norms and ontologies into agent organizations. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2004. LNCS, vol. 3346, pp. 181–198. Springer, Heidelberg (2005)
22. Esteva, M., Rosell, B., Rodriguez-Aguilar, J.A., Arcos, J.: AMELI: An Agent-based Middleware for Electronic Institutions. In: AAMAS 2004, pp. 236–243 (2004)
23. DeLoach, S., Oyenan, W., Matson, E.: A capabilities-based model for adaptive organizations. Auton. Agent Multi-Agent Syst. 16, 13–56 (2008)

A Systemic Approach to the Validation of Self–Organizing Dynamics within MAS

Jan Sudeikat and Wolfgang Renz

Multimedia Systems Laboratory,
Hamburg University of Applied Sciences,
Berliner Tor 7, 20099 Hamburg, Germany
Tel.: +49-40-42875-8304
{sudeikat,wr}@informatik.haw-hamburg.de

Abstract. Conceiving applications as sets of autonomous agents is a prominent approach to the construction of complex distributed systems. Particularly attractive are *decentralized* application designs that enable adaptive, robust and scalable applications by allowing agents to self–organize. Tools to the construction of self–organizing MAS, e.g. decentralized coordination strategies, catch increasing attention in MAS research. However, their purposeful utilization challenges current development practices. The intended non–linear macroscopic dynamics hinder top–down designs on the drawing board and corresponding development procedures rely on sequences of manual system simulation. In order to stimulate methodical development and facilitate the validation of complex MAS by simulation, we present a *systemic* approach to the *qualitative* validation of macroscopic MAS dynamics. Describing MAS as dynamical systems enables developers to formulate *hypotheses* on the intended macroscopic MAS behaviors that guide system simulations. We discuss and exemplify how to (1) derive systemic models as well as hypotheses from MAS designs, (2) infer appropriate simulation settings to their validation and (3) interpret the obtained results. In addition, work in progress on the automation of both system simulations and their interpretation is outlined.

1 Introduction

Agent–Oriented Software Engineering (AOSE) is a prominent development approach to the construction of complex distributed software systems. [1]. The utilization of *autonomous* and *pro–active* agents as a basic design and development metaphor is particularly attractive for applications that operate in dynamic environments. Flexible and scalable application architectures can be conceived, by

[1] Jan Sudeikat is doctoral candidate at the Distributed Systems and Information Systems (VSIS) group, Department of Informatics, Faculty of Mathematics, Informatics and Natural Sciences, University of Hamburg, Vogt–Kölln–Str. 30, 22527 Hamburg, Germany, jan.sudeikat@informatik.uni-hamburg.de

designing applications as *Multi–Agent Systems* (MAS). The MAS functionality results from agent interplay. Therefore, the quality of an agent–based application depends crucially on the effective *coordination* of agent activities.

A topic that calls increasing attention in MAS research is the development of *self–organizing* applications which exhibit system–wide adaptivity, due to the *decentralized* coordination of local, microscopic agent activities [2]. The term self–organization refers to physical, biological and social phenomena, where global structures arise from the local interactions of individuals (e.g. particles, cells, agents, etc.). A number of coordination strategies provide field–tested means to the establishment of these phenomena by embedding control loops among agent societies [3,4] (cf. section2). However, development efforts have to solve the dilemma how to design microscopic agent models and interactions to ensure the rise of intended globally observable properties. This gap between microscopic agent designs and the rising macroscopic structures hinders traditional, top–down oriented development processes and demands extensive system simulations in bottom–up procedures that understand development as sequences of experiments [5]. The comparison of simulation results with the expected macroscopic system dynamics is the major mean to ensure that the utilization of decentralized coordination strategies leads to the the intended system properties.

Recently, *system dynamics* modeling approaches have been applied to model and design self–organizing MAS (cf. section 3.2). These describe the structure of dynamical systems in terms of macroscopic system *state variables* and their *causal* relations. The transfer of these complex systems modeling techniques to MAS development facilitates descriptions of the dynamics that are intended by MAS designs. This approach is particularly attractive to the descriptions of self-organizing processes, as it highlights the presence of distributed control loops that steer self–organizing phenomena [3,4].

In order to stimulate methodical development approaches to self–organizing MAS and facilitate the unavoidable simulation cycles [5], we discuss how to derive description of the the expected, macroscopic MAS behavior from MAS designs and how to validate these *hypotheses* via system simulations. This practical approach supplements techniques to the validation of MAS (e.g. discussed in [6]) by checking how agent activities are *causally* related. It particularly addresses the validation of decentralized coordination strategies by checking that the intended effects on agent societies can be observed. In addition, the automation of simulations and their interpretation is outlined.

The remainder of this document is structured as follows. The next section outlines development approaches to self–organizing MAS. A systemic modeling approach to MAS is then outlined (section 3). The following section (4) discusses how these modeling notions can be used to describe *expectations* on macroscopic MAS dynamics, and how the validity of these expectations can be checked via (possibly automated) system simulations. These activities are exemplified in a case study (section 5). At the end, we conclude and give prospects for future work.

2 Building Self-Organizing MAS

A prominent approach to the utilization of self–organizing process in MAS is the *bionic* recreation of well–known dynamics. Nature–inspired *design metaphors* [7] as well as *decentralized coordination mechanisms* [8] provide field-tested coordination strategies that can be used to enforce macroscopic self-organizing phenomena. These result from *control loops* [3,9] that are distributed among agents and originate from the interplay of agent / environment interactions and collective behavior adjustments [4]. Prominent examples are MAS designs that coordinate via *stigmergy* [3] or *computational fields* [10]. Available coordination mechanisms have been classified and discussed according to their underlying computational techniques [2], the properties of resulting macroscopic phenomena [11] and their sources of inspiration.

The purposeful application of self-organizing process in MAS requires to [12]:

1. select an appropriate set of coordination mechanisms / metaphors,
2. refine an application design that maps the mechanism structure to the application domain,
3. implement the design, and
4. calibrate agent as well as environment parameters to adjust the intended behavioral regime of the software.

The first two steps are typically guided by heuristics and experience. In [11], the mechanism selection (1) is approached by associating mechanisms instances with macroscopic system properties. Mapping coordination mechanisms and design metaphors to application domains (2) requires their purposeful adjustment and redesign (e.g. discussed in [12]). In [13], a catalogue of environment mediated design metaphors (from [7]) has been extended with systemic models (cf. section 3) of the control loops that these pattern establish in MAS. These models explicitly describe the MAS behaviors that are enabled by the adoption of coordination strategies and have been applied to guide the design of metaphor combinations [13].

Simulation-based development procedures have been proposed to guide the remaining (3/4) development activities. Self-organizing properties elude from formal microscopic modeling [5]. Therefore, development teams usually revise prototype implementations to *qualitatively* evaluate that MAS designs lead to the intended behavior and to *quantitatively* tune implementation parameters. An experimental stance towards application development is composed of the iterative alternation between engineering and adaptation activities of software implementations [5]. A corresponding extension to testing activities in the Unified Process has been proposed that utilizes macroscopic system simulations [14]. However, the derivation of simulation settings and the interpretation of simulation results are manual activities. The *testability* of MAS dynamics, by specifying macroscopic behaviors and checking these expectations, has not yet been discussed.

3 Modeling Macroscopic MAS Dynamics

A number of organizational modeling approaches (e.g. reviewed in [15]) are available to MAS developers. The provided concepts and notations typically focus on static organizational settings, and the need for modeling approaches to the dynamics of MAS organizations has been recognized (e.g. see [15]).

Particularly, the development of adaptive, self–organizing MAS benefits from techniques to model the dynamic behavior of MAS. The transfer of concepts and notations from *system dynamics* [16] research adresses this challenge.

3.1 System Dynamics

System dynamics research approaches the examination of complex system behaviors by macroscopic modeling the *causal* structure of systems [16]. This system theoretic framework has been applied to various application domains, particularly in economic settings. The state of a system is described by a set of state variables. These variables quantify the system state. The mutual influences of state variables are described as *causal* relations. These relations connect state variables and describe how variable values are correlated, i.e. how changes in state variables effect connected system elements. These influence can be either positive or negative. An example for system state variables are savings accounts and interest rates. An interest rate positively influences, i.e. continuously causes an increase of, a savings account balance. More precisely, positive relations effect changes in similar directions. I.e. an increase in an originating variable causes an increase in connected system variables and an decrease in originating variables causes the decrease of connected variable values. Negative relations respectively invert effects, i.e. increase causes decrease and vice versa.

A prominent graph–based notation is the *Causal Loop Diagram* (CLD) [16]. In CLDs, system variables are denoted by nodes that are connect by arrows which indicate positive (+) or negative (-) causal relations. Links can form circular structures. When these are composed of an even number of negative links their influences add up to a so–called *reinforcing* (R) feedback loop that amplify perturbations on the system. An odd number of negative links enforce a *balancing* (B) behavior that impose self–correcting behavior, i.e. opposes disturbances.

Figure 1 (I) exemplifies the CLD notation by denoting the structure of the generic *Lotka-Volterra*[1] model that is commonly found in competitive animal populations. A population of *Prey* is constantly accrete by a certain rate (*Growth Rate*). The rate of growth depends on the population size, due to the possible offspring, and these relations form a reinforcing feedback loop. The population grow infinitely but preys are foraged by a population of *Hunters*. The number of hunters is constantly limited by a specific *Death Rate* that enforces a balancing feedback loop. Encounters of member of both populations (*Encounter Rate*) limit the number of preys (negative link) and increase the number of hunters (positive link) in the system. The number of encounters (positively) depends on the size

[1] http://mathworld.wolfram.com/Lotka-VolterraEquations.html

of both populations. This causal structure of four feedback loops is capable to exhibit an oscillatory behavior. The outlined CLD formalism is attractive for the description of complex systems, as it highlights the causal structure of systems and allows to relate these structures to their dynamic behavior. E.g. pattern of causal structures can be related to *behavioral modes*, i.e. pattern of timely behaviors.

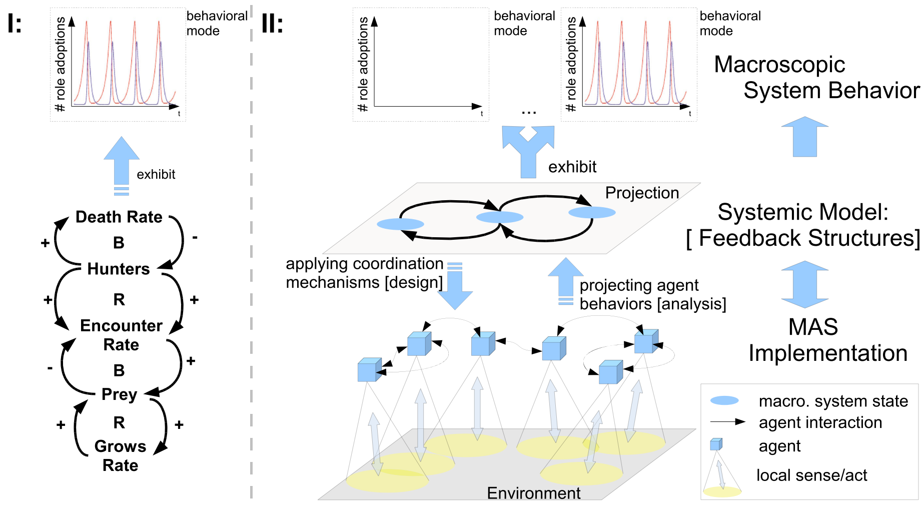


Fig. 1. I: An example CLD. II: MAS implementations can be projected into system dynamics models that describe the causal relations between macroscopic system states

3.2 System Dynamics for MAS

Here, we show how systemic models can describe MAS. Systemic, e.g. CLD–based, representations of MAS designs / implementations address the explicit description of the organizational dynamics within MAS. Agent–oriented modeling approaches [15] typically refer to *roles* and *groups* as abstractions on agent activity. Roles can be used to describe the normative agent behaviors that individual agents can commit to. The group concept allows to describe the formation of collaborations among individuals that share characteristics (e.g. see [15]). Both notions are typically applied within AOSE practices to describe the static semantics of MAS organizations in terms of role/group formations. Associated models [17] can be supplemented by describing the interactions between roles with the tailored sequence diagrams of *AUML* [18,19].

In agreement with mathematical modeling approaches (e.g. [20]), systemic models describe the macroscopic state of an MAS by the aggregate numbers of agents that play specific roles, respectively are members of specific groups. The transition between microscopic and macroscopic modeling levels is denoted in figure 1 (B). The microscopic agent designs comprise sets of roles / behaviors that agent can adopt, i.e. to execute certain tasks or to join / leave groups. Systemic models are constructed by by categorizing these local behaviors and projecting them to macroscopic system state variables (cf. figure 1, ii). Causal links indicate that specific role/group activities mutually influence the role changing, respectively group joining / leaving activities of other agents. These relations between state variables can be inferred by examining the detailed designs of role behaviors and how these interact with other agents, i.e. how agent interactions cause population members to change their allocations.For example, a hypothetical MAS may comprise agents that produce elements that are consumed by consuming agents. The detailed microscopic activities to manufacture elements can be categorized into a general system variable that describes the number of agents that are occupied with production processes. When consumers are enabled to request productions directly, their relation to producer agents is characterized by a positive causal relation, as increases in demand should normally lead to increases in production and decreases in demand should result in a decrease of requests. The derivation of systemic MAS models is exemplified in section 5.

This derivation of systemic models requires considerable manual modeling effort. Envisioning the causal structures of MAS designs allows the identification of feedback cycles that steer decentralized coordination of MAS as well as emergent phenomena. These phenomena may be explicitly intended, or be introduced unintended (e.g. discussed in [21]). Therefore, development teams may want to ensure the presence or absence of circular causalities in complicated designs with large numbers of agent types or roles. The here proposed modeling approach allows to anticipate organizational dynamics within MAS by identifying the behavioral modes that result from the identified causal relations and feedback structures. The validation of these anticipations is discussed in section 4.

Roles and groups are modeling abstractions that are independent of the agent architecture. Therefore, deliberative, reactive and hybrid agent architectures can be examined by this abstraction. Both AOSE design models (e.g. [18,19]) and MAS implementations can serve as microscopic models (figure 1 - B; bottom layer). E.g. in [22], the derivation of macroscopic system states has been approached by traversing the goal / sub–goal hierarchies of goal–directed agent designs.

4 Validating Macroscopic MAS Dynamics

When validating the results of system simulations in the testing stages of development procedures, e.g. as proposed in [14], one has to distinguish between *parametric*, *quantitative* validations of intended behavioral modes (e.g. the length of oscillation cycles) and the *non–parametric*, *qualitative* assurance that the intended causal relations are indeed exhibited [23]. Quantitative validations – by reaching agreements between dynamic models and agent–based applications – demands considerable modeling and parameter tuning effort (e.g. discussed in [24,25]) and is therefore of academic interest and seldom practiced in engineering projects. In order to support AOSE practices, we propose the utilization of qualitative validations that can be obtained routinely and allow to confirm that MAS agree to CLD abstractions, i.e. that the intended causalities manifest themselves in application simulations. This view–point especially supports the development of decentral coordinated MAS where examination of the effects of distributed control loops, that are embedded among agent societies (cf. section 2), is of vital importance.

4.1 Procedure

Systemic modeling notions provide means to anticipate application dynamics at design time [13]. Developers can use these to express the intended behavior of application designs. The here discussed strategy to the validation of MAS dynamics shows the agreement between systemic descriptions and application by checking whether the effects of individual causalities are observable in application simulations. The validation strategy (cf. figure 2)[2] comprises the (1) identification of causalities that are expected to the present, due to the coordination of agents and the selection of crucial influences (expressed as *hypotheses*) that are to be checked (*Conception*), the (2) to preparation of appropriate simulation settings, their execution and the measurement of system state values (*System Simulation*) and finally the (3) examination whether causalities manifest themselves (*Analysis*).

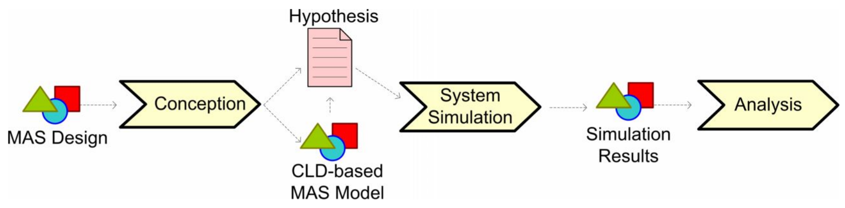


Fig. 2. Validation Procedure. Hypotheses on the macroscopic observable MAS behavior are extracted from MAS designs. These are validated by tailoring simulation settings and analyzing the obtained simulations results.

Conception. Initially, CLDs are derived from MAS designs to describe the intended application behavior. Developers identify the roles / behaviors that agents are capable to exhibit. These are classified to system state variables, i.e. CLD nodes. The causal links between these nodes are derived from the microscopic agent designs by tracing how *external*, e.g. environment–mediated or direct, interactions as well as agent *internal* reasoning influence the role / behavior changing characteristics of agents (cf. section 3.2). Then, individual relations can be selected for empirical validation.

[2] The notation follows the Software Process Engineering Meta–Model (SPEM): http://www.omg.org/technology/documents/formal/spem.htm

These relations are formulated as *hypotheses* on the expected dynamic behavior. Hypotheses follow the conceptual model given in figure 3 (A) and represent individual causal links and provide the information that are required to automate the parametrization of simulation settings and sweep parameter ranges (cf. section 4.2). A set of hypotheses (*Hypotheses*) is composed of individual expectations (*Hypothesis*) that are made up by *Dynamic* and *Parameter Space* components. *Parameter Space* components define fixed parameters (*Parameter*) and parameter ranges (*Range*) of variables. The *Dynamic* component defines the *observables* (macroscopic system states) which are expected to be causally linked.

System Simulation. In order to enable meaningful system simulations, that allow to observe the manifestation of causal relations, the MAS / simulation models need to be initialized appropriately. Initializations need to provoke that the distributed, possibly decentralized, coordination strategy triggers and steer the role / group changing behavior of agents. Two kinds of behavioral regimes can be distinguished (see [23]) that allow the observation of decentralized coordination. The impact of self–regulatory mechanisms can be observed (1) when systems *respond* spontaneously to environmental changes (so–called *responsive regime*) and also (2) when systems operate in *working regimes*, i.e. operations are continuously steered by the continuous intervention of feedback control mechanisms. Responsive regimes can be established by initializing the system in a way that the control mechanisms trigger immediately, e.g. by initializing a specific ratio of role assignments [23]. Then it can be observed how the MAS *responds* to the given configuration i.e. reorganizes its organizational structure (cf. figure 3; B). The observation of *working regimes* requires that the system is subject to a constant input that enforces continuous system adjustments, for example, by constantly or periodically reinforcing an environment property or role occupation (cf. figure 3; B exemplified in section 5.3).

Analysis. Simulations generate time series of state variable values, i.e. the counts of agents that exhibited specific roles / behaviors at given time points. The direct observation of causal influences in these measurements is complicated as the observations are likely to be subjects to perturbations and measurement noise. Instead, the presence of causal relations manifests itself as stochastic *correlations* of state variable changes. The mathematical interpretation of the causal links in the systemic MAS models indicates *correlations* between the time dependent behavior of system states, i.e. positive links indicate that changes in system states cause equally directed (delayed) changes in connected states. Negative causal links respectively indicate changes in opposite direction, i.e. negative correlations. The measurement of a correlation does not necessarily show causal influence. Correlation may also be side effects of unrelated influences. However, when causal relations are steering the behavior adjustments of individual agents, they are expected to manifest themselves in correlation of state variables. Therefore, calculating the correlation of system states allows to validate individual causal links. Equation 1 shows the *correlation function*, where x and y denote

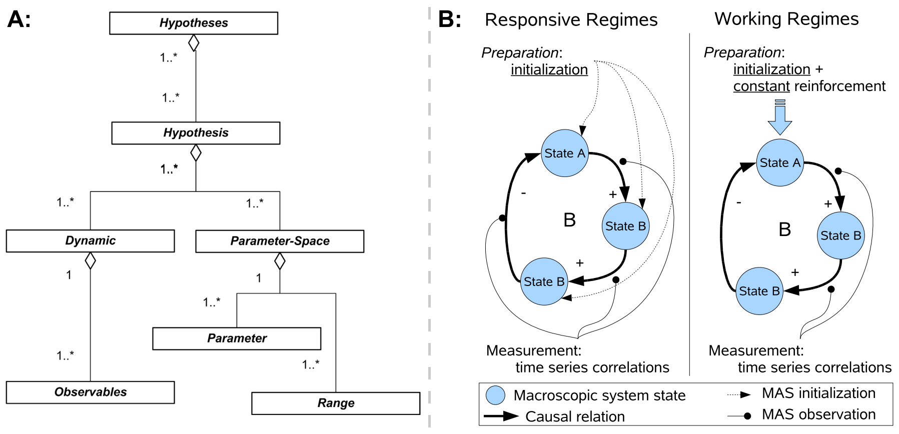


Fig. 3. A: The conceptual model of hypotheses specifications. B: Simulations require initialization and observation of system states. While appropriate initializations can provoke *responsive regimes*, a constant input is necessary to enforce *working regime* behavior.

the time series of system state occupations that develop with the time t. The terms $\overline{x}$, $\overline{y}$ denotes the (arithmetic) average values of x, respectively y.

$$C_{x,y}(\tau) = \frac{\sum (x(t) - \overline{x})(y(t+\tau) - \overline{y})}{\sqrt{(\sum x(t)^2/n - \overline{x}^2)(\sum y(t)^2/n - \overline{y}^2)}} \tag{1}$$

4.2 Automating Qualitative Validations

The initialization and execution of simulations as well as the mathematical treatment of the obtained data demands considerable manual effort. A prototype simulation execution environment has been conceived that automates simulations to check the the presence of causal relations by measuring the correlations of system state variables. An agent–based design[3] supports the distribution of simulation runs and facilitates parameter sweeps [26] by enabling concurrent simulation runs. Agents manage autonomously the user interaction, the validation of individual hypotheses and the execution of simulations.

The conceptual architecture is given in figure 4. *Users* examine MAS designs (cf. section 3.2) and provide the derived *hypotheses* (in XML language; according to the meta–model denoted in figure 3, A) as well as simulation models. *User* agents interact with the user, examine the provided hypotheses and delegate the validation of individual hypotheses to so–called *analysis* agents. These examine the correlations of monitored system state variables (cf. equation 1). These

[3] realized within the Jadex agent system (http://vsis-www.informatik.uni-hamburg.de/projects/jadex/)

agents use a *Numerical Computing Environment*[4] for the sake of efficiency. The specification of the hypotheses includes the parameter ranges that are to be examined. Therefore, analysis agents require data from parameter sweeps. The required simulation results are fetched from a *Networked Data Storage* that is used to save measurements. Measurements can be retrieved by their simulation parameter values. When simulation settings, i.e. specific parameter configurations, have not yet been examined, analysis agents request the required simulations from so–called *simulator* agents. These encapsulate the capabilities and computational resources for simulation execution. Simulators provide means to the *deployment*, *initialization*, *parametrization* and *measurement* of simulation environments and MAS prototype implementations. Simulation agents store the obtained results in the networked database for later use. Finally, the correlations of the observed state variable time series are presented to the user.

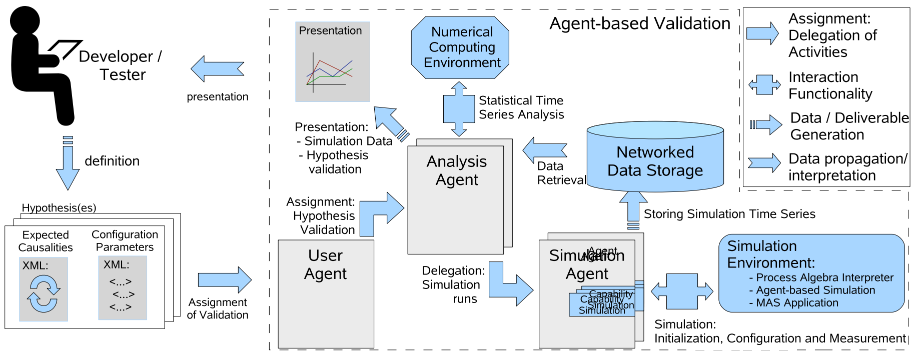


Fig. 4. A conceptual testbed architecture. Description see text.

5 Case Study: Intrusion Detection Dynamics

The application of the described validation approach (cf. section 4) is exemplified by the validation of a simplified intrusion detection system that follows the model given in [23]. We describe the application design and the derivation of a corresponding systemic model. A hypothesis is derived and validated by showing that the system simulation exhibits the correlations that are indicated by the systemic model.

5.1 A Simplified Intrusion Detection System

The conceptual design of the immune system inspired intrusion detection system (e.g. as discussed in [27]) is given in figure 5 (i). A *Proemetheus* [28] *System Overview Diagram*[5] denotes three agent types. A network of hosts is to be

[4] e.g. using Scilab (http.://www.scilab.org) via the the so–called javasci interface.
[5] created with the *Prometheus Design Tool* (PDT): http://www.cs.rmit.edu.au/agents/pdt/

protected from malicious *intruder* agents that manifest themselves by executing *Malicious Activities*. These activities are perceived (*Perceive Intruder*) by so–called *Searcher* agents that wander the network and inspect hosts. When infections are identified, these agents communicate (*Notification*) the presence of Intruder agents to *Remover* agents that are capable of disposing Intruders (*Removal* activity). Notifications are expected to be distributed via computational *pheromones* [3], i.e. the network is resembled by a virtual, spatial environment where patches represent hosts. Searchers release markers on patches to indicate the presence of intruders. These markers evaporate and diffuse [3]. Due to the diffusing markers, Removers get notified of infections. Upon removal (*removed*) Intruder agents stop functioning. A detailed discussion of the simulation setting and implementation can be found in [23].

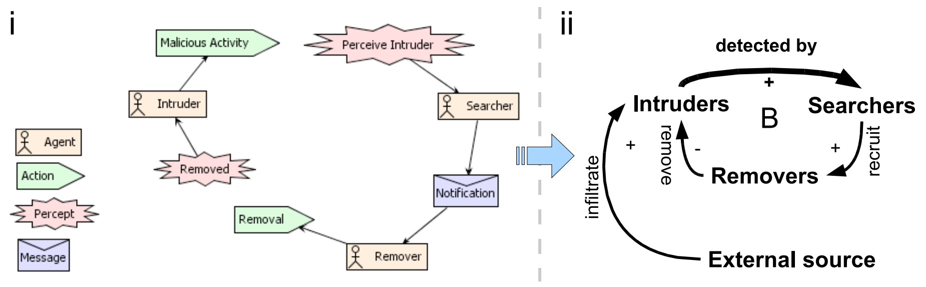


Fig. 5. The simplified intrusion detection model, following [23]. A Prometheus System Overview Diagram(i) and the description (CLD) of the macroscopic behavior (ii).

The here proposed validation of macroscopic MAS causalities is based on statistics, i.e. the measurement of time series correlation (cf. section 4.1). In order to demonstrate the practicability of the procedure and tool support, the here presented examination addresses the validation of simulations with high stochastic noise. Following the direction of [29] we use *stochastic process algebra* (SPA) models to resemble the given MAS as a set of independently executing processes that communicate information via synchronous channels.[6] The utilized SPA model (in stochastic π-calculus; see [30] for a detailed presentation and application examples) denotes the activities of agent types as algebraic formulated processes. The duration of agent activities is resembled by exponential distributed delays, and channel–based communication are controlled by exponential distributed channel interaction rates [31] that resemble environmental influences on agent activity execution, e.g. the diffusion of pheromones. The simulation results resemble the described MAS operation, but generate fluctuating measurements of agent operations.

[6] utilizing the stochastic π-calculus algebra and the freely available SPiM simulation engine: http://research.microsoft.com/~aphillip/spim/

5.2 Conception

Intruder, Searcher and Remover agents are classified into two roles. They are either *active* or *inactive*. Intruders are active by default and get deactivated by their removal from the system. Searchers wander the environment by default and get activated by the identification of infections. Upon activation, they start to release digital pheromones to notify removers. Remover agents are by default inactive and get activated by perceiving the presence of intruders, i.e. pheromones. After successful removal, Searchers and Removers are deactivated and continue with their default behaviors. Figure 5 (ii) shows a CLD that represents the expected causalities between agent activities. The nodes represent the numbers of activated agent types. When the searching algorithm and detection mechanism of searcher agent functions effectively, the presence of intruder agents will be noticed and therefore, causes the activation of Searchers. Therefore, both agent types are positively–liked (*detected by*), i.e. an increase of intruders in the system is expected to cause an increase in the amount of activated Searchers. When the amount of Intruders is decreasing, more Searchers should be occupied with their (default) searching role, thus the number of activated Searchers should decrease. The utilized mechanisms for the notification of Removers ensures the positive link between activated Searchers and Removers (*recruit*). The distribution of pheromones leads to the activation of Remover agents. Since the pheromones evaporate, the activation of removers is decaying as soon as the reinforcement of pheromones (by activated Searchers) stops. Removers are negatively linked to intruders since the activation of Removers leads to the removal of Intruders, i.e. The number of Intruders is lowered by the activation of Removers. For the sake of brevity, only the validation of the positive causal link between intruders and searchers (cf. figure 5, ii - bold arrow: *detected by*) is demonstrated. While this influence is build–in the simulation model (detection channel), the direct observation of this relation is in the generated time series complicated by the stochastics of process execution. A corresponding hypotheses has been defined to guide the automation of simulation runs (cf. section 4.2).

5.3 Simulation

In order to validate that intruders are reliably detected and are therefore marked for removal, a *working regime* (cf. section 4.1) is established in the simulation model. An external source of intruders has been added that generates a constant reinforcement of intruders (cf. figure 5, ii). After initialization with 100 intruders, 50 searchers and 25 removers, the system reaches detection rate–dependent, equilibrated agent densities that exhibit stationary correlations.

5.4 Analysis

Figure 6 (i–iii) show the simulation results. The numbers of activated Intruders and Searchers have been measured and their correlations has been computed (cf. equation 1) for successive simulation runs. The simulation environment (cf. section 4.2) allows to automate the required simulations / computations and facilitates the examination of the systematic behavior, i.e. rate–dependence, of the observed correlation. The correlation maxima indicate the expected causal dependence. The causality is associated with a time delay that is quantified for different detection rates. The figure (ii) shows a *master plot* of the cross–correlation functions for different channel interaction rates (ranging from 0.4 to 40) that were shifted in time and amplitude. The time shift that allows to normalize the correlation maxima to one is described by a logarithmic fit (iii), while the corresponding shifts of the correlation amplitudes (iv) are described by an algebraic fit.

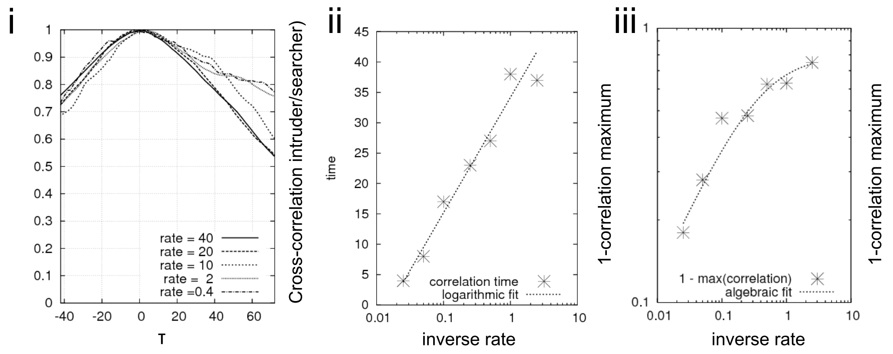


Fig. 6. Simulation results of the simplified IDS dynamics: (i) The cross–correlation between intruders and searchers exhibits rate dependent maxima at finite correlation times. (ii) The time shifts are given by the rate–dependent correlation times. (iii) The amplitude shifts are given by the rate–dependent correlation maxima.

6 Conclusions

In this paper, the utilization of system dynamics modeling notions for the validation of MAS has been proposed. These models facilitate the description the intended macroscopic MAS behaviors. We showed how to systematically validate the agreement between MAS implementations / simulation models and systemic MAS models. Agreements between systemic expectations and empiric system observations indicate the proper use of MAS coordination strategies.

Future work will examine *analytic* and *constructive* usages of dynamic MAS models to supplement MAS development practices. Work in progress also comprises the automation and interpretation of MAS simulations by a MAS–based simulation execution environment. Besides, the automation of quantitative validations and parameter tuning efforts is a topic of future work.

Acknowledgments

One of us (J.S.) would like to thank the *Distributed Systems and Information Systems* (VSIS) group at Hamburg University, particularly Winfried Lamersdorf, Lars Braubach and Alexander Pokahr for inspiring discussion and encouragement.

References

1. Jennings, N.R.: Building complex, distributed systems: the case for an agent-based approach. Comms. of the ACM 44(4), 35–41 (2001)
2. Serugendo, G.D.M., Gleizes, M.P., Karageorgos, A.: Self–organisation and emergence in mas: An overview. Informatica 30, 45–54 (2006)
3. Parunak, H.V.D., Brueckner, S.: Engineering swarming systems. In: Methodologies and Software Engineering for Agent Systems, pp. 341–376. Kluwer, Dordrecht (2004)
4. Sudeikat, J., Renz, W.: Building Complex Adaptive Systems: On Engineering Self–Organizing Multi–Agent Systems. In: Applications of Complex Adaptive Systems, pp. 229–256. IGI Global (2008)
5. Edmonds, B.: Using the experimental method to produce reliable self-organised systems. In: Brueckner, S.A., Di Marzo Serugendo, G., Karageorgos, A., Nagpal, R. (eds.) ESOA 2005. LNCS (LNAI), vol. 3464, pp. 84–99. Springer, Heidelberg (2005)
6. Sudeikat, J., Braubach, L., Pokahr, A., Lamersdorf, W., Renz, W.: Validation of bdi agents. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2006. LNCS (LNAI), vol. 4411, pp. 185–200. Springer, Heidelberg (2007)
7. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. J. Syst. Archit. 52(8), 443–460 (2006)
8. DeWolf, T., Holvoet, T.: Decentralised coordination mechanisms as design patterns for self-organising emergent applications. In: Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications, pp. 40–61 (2006)
9. DeWolf, T., Holvoet, T.: Using uml 2 activity diagrams to design information flows and feedback-loops in self-organising emergent systems. In: Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems, EEDAS 2007 (2007)
10. Mamei, M., Zambonelli, F., Leonardi, L.: Co–fields: A physically inspired approach to motion coordination. IEEE Pervasive Computing 03(2), 52–61 (2004)
11. DeWolf, T., Holvoet, T.: A taxonomy for self-* properties in decentralised autonomic computing. In: Autonomic Computing: Concepts, Infrastructure, and Applications (2006)
12. Sudeikat, J., Renz, W.: On the redesign of self–organizing multi–agent systems. International Transactions on Systems Science and Applications 2(1), 81–89 (2006)
13. Sudeikat, J., Renz, W.: Toward systemic mas development: Enforcing decentralized self–organization by composition and refinement of archetype dynamics. In: Weyns, D., Brueckner, S.A., Demazeau, Y. (eds.) EEMMAS 2007. LNCS, vol. 5049, pp. 39–57. Springer, Heidelberg (2008)
14. DeWolf, T., Holvoet, T.: Towards a methodolgy for engineering self-organising emergent systems. In: Proceedings of the International Conference on Self-Organization and Adaptation of Multi-agent and Grid Systems (2005)
15. Mao, X., Yu, E.: Organizational and social concepts in agent oriented software engineering. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 1–15. Springer, Heidelberg (2005)
16. Sterman, J.D.: Business Dynamics - Systems Thinking an Modeling for a Complex World. McGraw-Hill, New York (2000)
17. Odell, J.J., Parunak, H.V.D., Brueckner, S., Sauter, J.: Temporal aspects of dynamic role assignment. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 201–213. Springer, Heidelberg (2004)

18. Odell, J., Parunak, H.V.D., Bauer, B.: Extending uml for agents. In: Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence (2000)
19. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: An organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
20. Lerman, K., Galstyan, A.: Automatically modeling group behavior of simple agents. In: Agent Modeling Workshop, AAMAS 2004 (2004)
21. Mogul, J.C.: Emergent (mis)behavior vs. complex software systems. Technical Report HPL-2006-2, HP Laboratories Palo Alto (2005)
22. Sudeikat, J., Renz, W.: Monitoring group behavior in goal–directed agents using co–efficient plan observation. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 174–189. Springer, Heidelberg (2007)
23. Sudeikat, J., Renz, W.: On expressing and validating requirements for the adaptivity of self–organizing multi–agent systems. System and Information Sciences Notes 2(1), 14–19 (2007)
24. Axtell, R., Axelrod, R., Epstein, J.M., Cohen, M.D.: Aligning simulation models: A case study and results. Computational & Mathematical Organization Theory 1(2), 123–141 (1996)
25. Wilson, W.: Resolving discrepancies between deterministic population models and individual–based simulations. The American Naturalist 151, 116–134 (1998)
26. Brueckner, S.A., Parunak, H.V.D.: Resource-aware exploration of the emergent dynamics of simulated systems. In: AAMAS 2003: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pp. 781–788. ACM Press, New York (2003)
27. Liu, J., Tsui, K.: Toward nature-inspired computing. Commun. ACM 49(10), 59–64 (2006)
28. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons, Chichester (2004)
29. Gardelli, L., Viroli, M., Omicini, A.: On the role of simulations in engineering self-organising mas: The case of an intrusion detection system in tucson. In: Brueckner, S.A., Di Marzo Serugendo, G., Hales, D., Zambonelli, F. (eds.) ESOA 2005. LNCS (LNAI), vol. 3910, pp. 153–166. Springer, Heidelberg (2006)
30. Blossey, R., Cardelli, L., Phillips, A.: A compositional approach to the stochastic dynamics of gene networks. In: Priami, C., Cardelli, L., Emmott, S. (eds.) Transactions on Computational Systems Biology IV. LNCS (LNBI), vol. 3939, pp. 99–122. Springer, Heidelberg (2006)
31. Priami, C.: Stochastic π–calculus. Computer Journal 6, 578–589 (1995)

Using and Extending the SPEM Specifications to Represent Agent Oriented Methodologies

Valeria Seidita[1], Massimo Cossentino[2], and Salvatore Gaglio[1,2]

[1] Dipartimento di Ingegneria Informatica, University of Palermo, Palermo, Italy
{seidita,gaglio}@dinfo.unipa.it
[2] Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche Palermo, Italy
cossentino@pa.icar.cnr.it

Abstract. Situational Method Engineering used for constructing ad-hoc agent oriented design processes is grounded on a well defined set of phases that are principally based on reuse of components coming from existing agent design processes; these components have to be stored in a repository. The identification and extraction of these components could take large advantages from the existence of a standardized representation of the design processes they come from. In this paper we illustrate our solution based on SPEM 2.0 specifications for modelling agent design processes and extending them when necessary to meet the specific needs we faced in our experiments.

1 Introduction

Our research is focussed on the field of Situational Method Engineering (SME) [2][8][9][10] for the construction of ad-hoc multi agent systems design processes. Applying Situational Method Engineering requires executing a well defined set of phases [6][7][17]: Process Requirements Specification, Process Fragments Selection and Process Fragments Assembly.

SME is based on the reuse of components coming from existing design processes, the so called method fragments or simply fragments; the request for reusable fragments leads to the need for a repository containing standardized fragments that could be easily selected and assembled in new design processes (i.e. methodology; this term is commonly used in the agent community and in the SME one, for avoiding confusion in this work we consider it as a synonymous of design process or process).

Since the repository is composed of fragments coming from existing design processes, its construction cannot be done without considering: the knowledge of a set of existing processes, their standard description through a standard notation and a precise definition of the fragment notion and of the process itself.

We decided to use SPEM (Software Process Engineering Metamodel) 2.0 [11], both for design process and fragments representation; it is an OMG standard and it is based on a metamodel containing three main elements: activity, work product and process role. We found the use of SPEM very promising and suitable for our purposes; in our previous works [5][16] we identified and defined the main elements a design process and a fragment are composed of, these elements can be easily represented using the SPEM metamodel main elements.

Besides, in the agent oriented context, according to our view, a process is devoted to design a MAS model whose elements are represented in the work products resulting from the enactment of a specific activity. A MAS model element is an instance of a MAS metamodel element; the MAS metamodel represents the structure of the system that is being built with the specific design process.

The key point of our approach consists in using an ontological metamodel providing the right semantics to the element of the domain we are dealing with [3][15] (this metamodel will for instance make use of relationships that are typical of the ones used in ontologies). The MAS metamodel is one of the most important factors of our approach, as it is not present in the SPEM specifications we decided to extend these specification.

In this paper we present how we use SPEM, its elements and diagrams, for representing a design process and how we extended it by adding elements and diagrams to SPEM specification, in order to meet our needs.

The paper is structured as follows: in section 2 an overview on the Situational Method Engineering approach for creating new agent oriented design processes is given; in section 3 the main SPEM elements we use are illustrated, the needed extensions are justified and motivated and an example on applying SPEM is provided; finally in section 6 some conclusions are drawn.

2 The Formal Description of a Design Process

The construction of a new design process following Situational Method Engineering principles is based on three main phases [6][13]: Method Requirements Engineering, Method Design and Method Construction.

Our approach for applying SME in the agent oriented context is based on these three phases too but it is specialized for the agent context, as shown in Figure 1; here we sketch the whole process a method designer has to carry out in order to construct a new agent oriented design process.

The first step a method designer must undertake is to create a repository of fragments starting from those extracted from a set of existing design processes and/or constructing new ones from scratch. For this aim he needs the set of existing design processed to be well defined, in a standardized fashion, in order to easily and quickly identify the portions of process devoted to become fragments.

During the Process Requirements Analysis activity, the method designer considers inputs coming from the development context (tools, languages, available skills, etc.) and the type of problem he has to solve. These inputs are used to define the process life cycle (that establishes the structure the designer has to follow during process fragments assembly activity), the MAS metamodel concepts and the other process elements (available stakeholders, required activities or work products) that are used for selecting fragments from the repository.

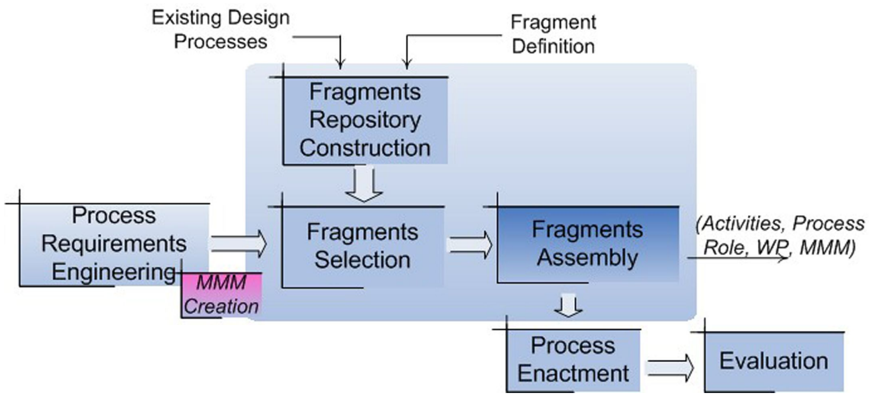

Existing Design Processes
Fragment Definition
Fragments Repository Construction
Process Requirements Engineering
MMM Creation
Fragments Selection
Fragments Assembly
(Activities, Process Role, WP, MMM)
Process Enactment
Evaluation

Fig. 1. Our Approach for Creating an Agent Design Process

It is worth noting the fundamental difference between the MAS metamodel that will be instantiated in the actual agent system during design and the process (or method fragment) metamodel that will be used to define the design process.

The output of the process requirements analysis contributes to the selection of process fragments and to their assembly; once the process is created, it can be enacted and then evaluated for eventually iterating the whole construction process.

In this work we want to point out our attention to the fragments repository construction, for which we need two elements: a set of existing design processes used for extracting fragments to be stored and a specific definition of design process. A specific definition of fragment is also needed for enabling the method engineer to correctly describe the existing design processes.

As regards the definition of design process and fragment we consider a design process as the set of activities to be performed in order to produce an output, the way of performing some activities (guidelines or techniques), and the resources and constraints this requires. In [5] we gave a definition of multi-agent system design process and of fragment (we call it *process fragment*). In Figure 2 we show the main elements we use for describing the agent design processes; these elements are the base for the design process fragmentation.

A design process is composed of activities, each activity is performed by a process role that is responsible for one or more work products that are structured by a work product kind representing a specific category, for instance, text document, code and so on.

A design process is devoted to design a MAS Model that is composed of MAS model elements each of which can be represented in one or more work products; a MAS model element is an instance of a MAS metamodel element so in each work product there is a correspondence with one or more MAS metamodel elements.

A process can be decomposed into (process) fragments that are self-contained pieces of the whole process, with all the elements characterizing the process

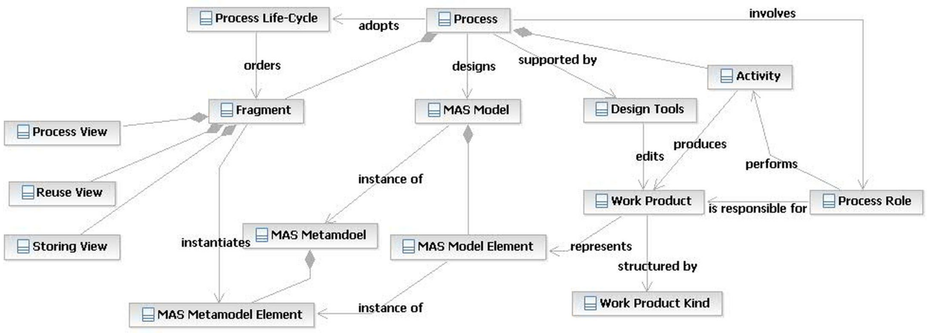


Fig. 2. The Agent Oriented Design Process Definition

itself (activity, process role, work product and MAS metamodel element) and that instantiate one or more MAS metamodel elements described in the work product(s) resulting for the fragment itself.

In our approach we assume that each fragment has to deliver at least one work product, thus, basing on this hypothesis and on the proposed fragment definition, we can say that our fragment extraction activity is work product driven.

Starting from the aforementioned definition of design process we decided to follow a top-down approach for a clear description and an easy retrieval of the main elements of a process: starting from highest level activities, we decomposed them (and the corresponding fragment) down to the atomic steps that compose the work to be done; at each level of detail, we report the produced work products (with their work product kind) and the specific process role that perform/assist the work and above all the description of the MMM element that is defined/refined/quoted in each work product.

Because of our formalization needs we decided to adopt SPEM 2.0 (Software Process Engineering Metamodel) Specification; it is based on the idea that *"a software development process is a collaboration between abstract entity called process role that performs operations called activities on tangible entities called work products"* [11]. This well defined conceptual model allows to represent every kind of process lifecycle (iterative, incremental, waterfall and so on), SPEM in fact is composed of a breakdown structure allowing to represent all design processes.

In the next section we will describe which elements and diagrams we use of this specification and for each of them we will illustrate the related definition (from [11]).

3 Using SPEM for Representing Agent Oriented Design Process

Software Process Engineering Metamodel (SPEM) [11] is a meta-modelling language used for the description of development design processes and their components.

The SPEM 2.0 presents a metamodel structured in seven main packages; only three of them will be illustrated in this paper in order to justify their specific use for our purposes, they are *Process Structure*, *Method Content* and *Process With Method* Packages:

- The *Method Content* package contains all the elements for creating a set of reusable methods, its aim is to illustrate which are the goals that a method has to reach, which resources are used and which roles are involved.
- The *Process Package* is composed of the main elements for modelling a process: Activities, nested in a breakdown structure where the performing Role classes and the input and output Work Product classes for each activity are listed.

These elements are used to represent a high-level process that when instantiated on a specific project takes the method content elements and relates them into partially-ordered sequences that are customized to specific types of projects.

For our purposes and at a first level of abstraction, a process, through the Process Package elements, can be represented in its general structure without any reference to a specific project and without detailing the inner content description of each activity.

It is worth to note that in SPEM 2.0 the concepts of "*Method*" and "*Process*" have a specific meaning that allows their use respectively for representing and modelling the fragment and the design process.

SPEM 2.0 presents two levels for process representation: Method is considered at an higher abstraction level, where there is no reference to a specific project (and above all it is considered as an auto consistent portion of process) whereas a Process is the concrete representation of a specific development situation.

For all these reasons we found SPEM 2.0 suitable both for our top-down decomposition/representation of a design process and for the representation of each fragment. We can use the *Method Content* Package elements to represent a (process) fragment and the *Process With Method* Package elements to represent an existing design process; note that a development process, in our approach, is composed of process fragments (see section 2).

Finally in the *Process with Method* Package the central element is the BreakdownElement; in SPEM 2.0 processes are represented with a breakdown structure of Activities that nest BreakDown Elements; they are generalization of any type of Process Element such as other Activity instances, Task Uses and Role Uses. Therefore Activity represents a basic unit of work within a Process as well as a Process itself.

Role Uses represent the performer of a Task Uses and defines a set of related skills, competencies and responsibilities of a set of individuals and *Task Uses* define the unit of work that is performed by Roles; a Task Use has a clear purpose in which the performing roles achieve a well defined goal. *Work Product Uses* are the artifacts, produced, consumed or modified by Task Uses[1]; Roles use Work Products to perform Tasks and produce Work Products in the course of performing Tasks.

[1] From now on, for the sake of brevity, Task and Task Use, Role and Role Use, Work Product and Work Product Use will be indifferently used, with the same meaning.

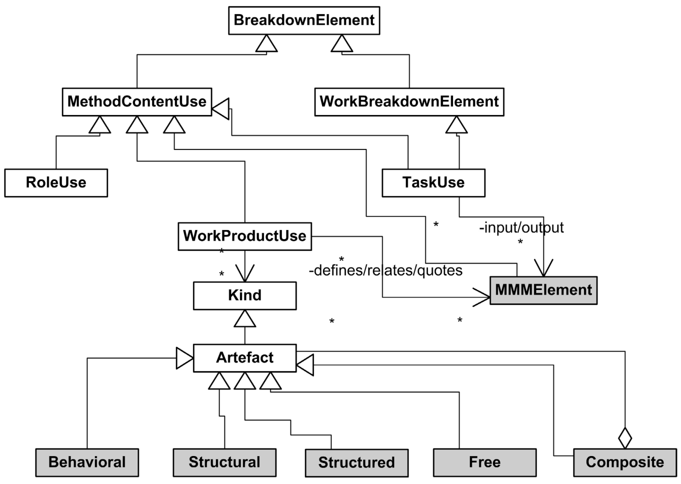


Fig. 3. The Proposed Extension to Process With Method Package Metamodel

The *Process With Method* Package contains the same elements of the process metamodel we presented in section 2, the Activity can be related/mapped to Task Use, The Process Role to Role Use and obviously the Work Product to Work Product Use. These elements together with the rationale of the work breakdown structure are sufficient enough for describing a complete design process for our purposes following a top-down approach, from the higher level definition of the work to be done until the details of each task with the roles performing it and the artifacts produced.

SPEM provides other two useful elements for grouping a set of activities under a common application theme, they are the *Process Component*containing one Process represented by an Activity and a set of Work Product Ports that define the inputs and outputs for the Process Component, and the *Phase* that represents a significant period in a project, with milestone or set of Deliverables.

With all the discussed elements a whole process can be divided into process components that groups the activities under a common theme, set of activities are also grouped into phases that impose specific milestones to the work to be done. Techniques, methods and guidelines for each activity are given for tasks that are performed by roles and consumes/produces (has input/output) work products.

4 Extending SPEM Specifications

In the previous section we saw which elements of SPEM metamodel packages we use for our purposes of providing a standard representation for agent oriented design processes. In this section we illustrate the motivation that led us to extend SPEM.

We had to deal with three specific factors that had a direct consequence in the extension of SPEM specifications: the MAS Metamodel element and the Work Product Kind that are elements of our process metamodel not provided by SPEM Packages and the existing dependencies among work products.

Regarding the MAS metamodel element, its definition is the core of the work done in each portion of process resulting in the delivery of a work product; as section 2 illustrates, using a specific design process for developing an agent system means to instantiate its metamodel, this instantiation results in a precise set of actions that we identified to be done on each MMM element; a designer can in fact

- **define** an element (i.e. instantiate it) thus establishing its properties and/or attributes; the resulting instantiated element is, of course, reported in the work product produced by the fragment the designer is executing
- similarly, the designer can **relate** a MMM element to other elements or
- simply **quote** it for introducing relationships among work products.

SPEM 2.0 does not provide means for representing actions to be made, in each activity, on the MMM elements.

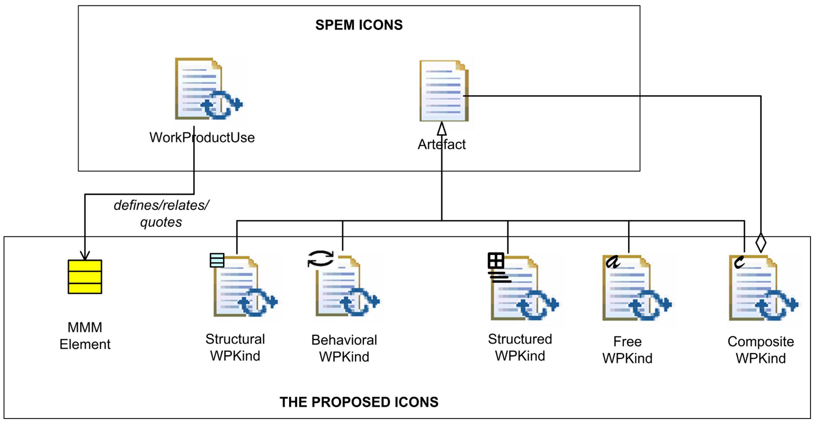


Fig. 4. The Proposed Icons

An important factor in the process representation is also modelling work products dependencies; SPEM specifications provide a way for modelling dependencies through the so called Work Product Dependency Diagram.

The Work Product Dependency diagram supplies way for representing three kinds of dependency relationships among work products: Composition, Aggregation and Dependency. The first expresses that a work product is a part of another one, the second indicates that a work product uses another work product and finally the third indicates that a work product depends on another one.

In our case a well specific dependency among a work product and another exists, it is due to the relationships among elements in the MAS metamodel; for instance let us consider an hypothetical requirement elicitation phase of a given process, we could think that this portion of process is represented by a metamodel where the concept of actor is related to that of scenario. Now, let us further suppose that the designer performing this phase begins his work by defining the concept of actor and produces a document listing all the actors he has identified. When the designer has to define the concept of scenario, because it is related to the concept of actor, he has to look at the last document he created, he cannot proceed without knowing all the defined actors; in so doing all the relationships among metamodel elements are reflected upon a precise dependency among the work products that act upon them.

In order to represent such situations, we need a diagram reporting both the dependencies among work products (and this is provided by SPEM) and the correspondence between each work product and the MMM elements, this second point led us to the creation of a specific diagram.

The SPEM Work Product Kind allows to represent a work product when it: *i)* is an intangible one or it is not formally defined (in this case it is of the kind: Outcome), *ii)* it aggregates other work products (the kind is Deliverable) and *iii)* it defines a tangible work product consumed, produced or modified by a task (the kind is Artifact).

In our work we defined five kind of work products [16] as a result of our need for adequately storing process fragments in our repository; the defined kinds on work products are:

1. Behavioural, it is a graphical kind of work product and is used to represent the dynamic aspect of the system (for instance a sequence diagram representing the flow of messages among agents along time);
2. Structural, it is also a graphical kind of work product and is used for representing the static aspect of the system, for instance a UML class diagram;
3. Structured, it is a text document ruled by a particular template or grammar, for instance a table or a code document;
4. Free, it a document freely written in natural language.
5. Composite, this work product can be made by composing the previous work product kinds, for instance a diagram with a portion of text used for its description.

These definitions together with that proposed by SPEM allow us to consider the work product kind we use as a specialization of the SPEM Artifact Work Product Kind (Figure 3).

Therefore for applying our design process definition we made some specific extensions to the Process With Method Package by adding some elements; in Figure 3 we show the portion of Process With Method metamodel that we extended; the white elements are all the pre-existing SPEM ones whereas the gray elements are newly introduced one.

To sum up, we added five elements and for each of them the related icon was created (see Figure 4); the relationship between Work Product Use and

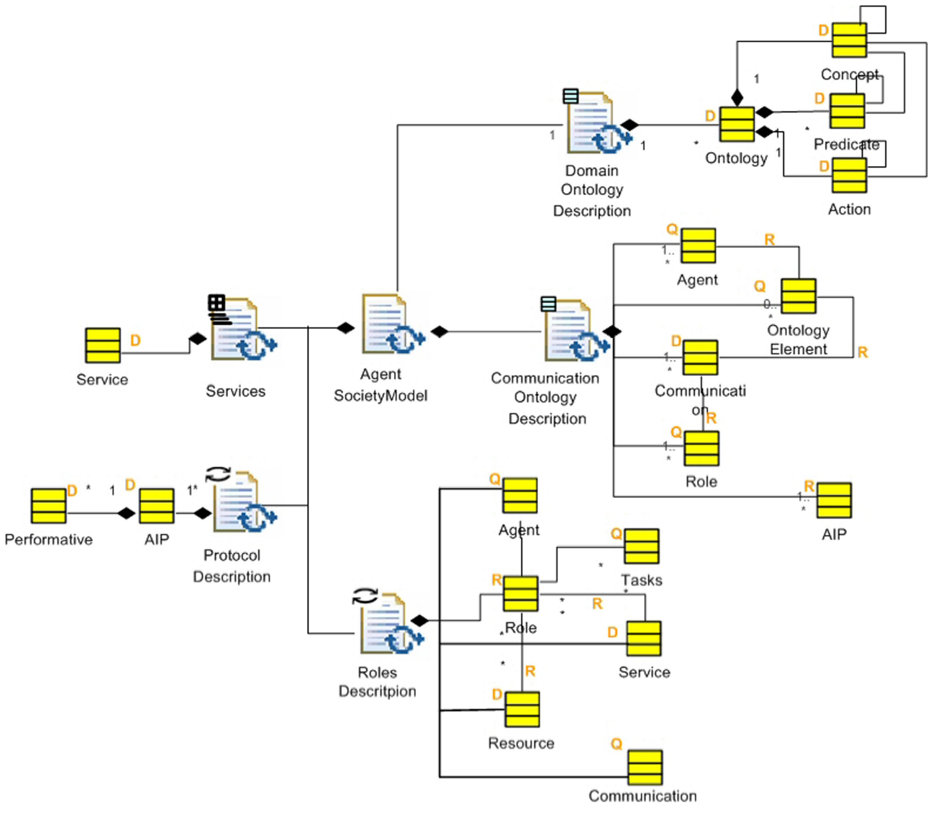


Fig. 5. The Correspondence among Work Product and MMM elements

MMMElement found its realization in a new diagram that represents the correspondence between each work product and the MMM elements it defines/ relates/quotes.

Figure 5 shows the kind of diagram we created in order to represent dependencies amog Work Product through MMM elements and shows the artefact produced during the PASSI Agent Society phase; it is composed of work products of three kinds (structural, behavioural, structured) and for each of them all the MMM elements they work on are shown, the letter indicates the specific action that is made on the MMM element: **D** stands for Define, **R** for Relate and **Q** for Quote.

Referring to the Agent Society phase that one of the Process Component elements of the PASSI process, as it will be seen in the next section, using the presented kind pf diagram we can see that it results in the Agent Society model, modelled through a Work Product Use composed by two structural work products, two behavioral ones and one structured.

One of the greatest advantages of using the diagram shown in Figure 5 is to have a complete vision on all the metamodel elements that are defined/related/

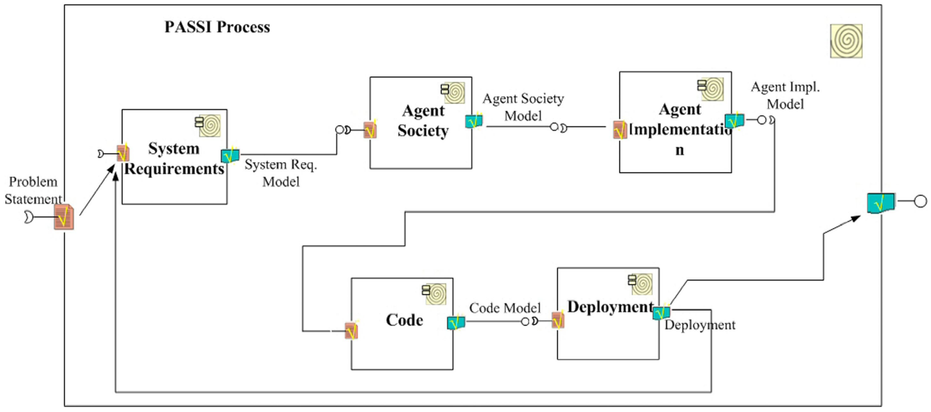


Fig. 6. The PASSI Process Component Diagram

q-uoted in each work product delivered from each process activity. Mind that the proposed SPEM representation aims at identifying reusable fragments from existing design processes and that in our approach each process fragment is devoted to instantiate (hence define) at least one MMM element. Such a view on the existing dependencies among MMM elements and work products allows us to quickly identify the possible fragments (how the fragments extraction is carried out is not the focus and is not detailed in this paper).

5 Representing PASSI with SPEM 2.0

Our representation of a design process follows a well specific method, it is carried out in a top-down fashion in order to reach the right level of granularity allowing the extraction of process fragments.

Therefore a process is represented, in a first time, as a package of components through the *Process Component Diagram* that allows to represent all the portions of a design process with the needed input and output, this diagram models the design process at a high level of detail.

In Figure 6 it can be seen, for instance, the Process Component Diagram related to PASSI [4], it is composed of five components, each of them representing a phase, a portion of work for which a specific outcome and milestones can be identified and represented in this diagram.

The second step is to detail the work done in each component through a SPEM Activity Diagram, Figure 7 shows all the activities nested in the Agent Society component, the activities sequence is indicated through the «predecessor»stereotype (the pointed activity is the predecessor of the other one). Each activity is composed of tasks, roles performing the task and input/output work products, no tasks sequencing is necessary at this step, it is only useful to give the idea of the set of elements an activity is composed of (in the figure only the Role Description activity is shown for space reasons).

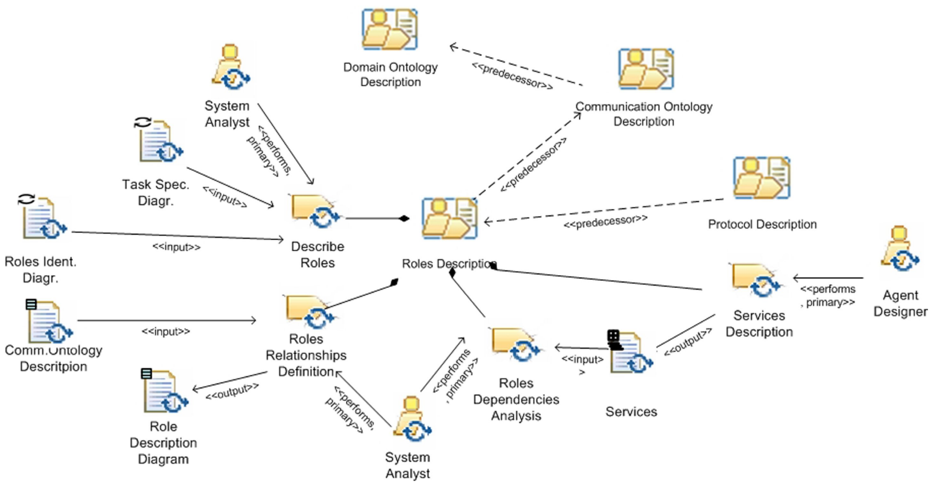


Fig. 7. The PASSI Process Activity Diagram

When complete, this diagram is full of information that could be accompanied by definition, explanation and so on, but it can result too huge, in alternative another diagram can be produced where only activity and input/output work products are shown.

This level of detail is sufficient in order to identify all the process fragments that can be extracted from a design process.

For a detailed documentation of the whole design process other diagrams have to be produced; these diagrams allow to model and document all the techniques, the methods and the guidelines involved in each task, i.e. the dynamic part of each portion of process, for space reasons we do not specify this part in this paper.

As already said, our process fragment extraction is work product-oriented [5], in the sense that we look at a work product and extract from the whole process only the portion of work delivering the selected work product.

For instance looking at the Role Description we can see that there are two outputs, Services and Role Descritpion diagram; then from Figure 5 (this is the final digram we draw when we model a design process) we can identify the MMM elements these work products act on and so we could decide to extract a fragment dealing with the concept of role and delivering the Role Description diagram and/or one dealing with the concept of service; of course it is also possible to identify a fragment delivering both the work products.

The novelty of the presented work is principally the possibility of easily retrieving a set of reusable fragments starting from the representation in a standard way of available design processes and then, once the fragments are identified and extracted, they can be documented in a similar standard way.

In our approach, in order to represent fragments, we also need elements such as activities, roles, work product and MMM elements and the way we indicate to use SPEM 2.0 specification and our extension will serve this scope.

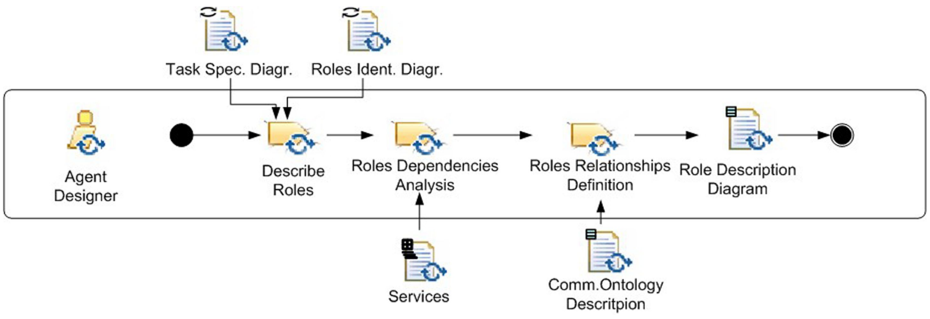


Fig. 8. The ”Roles Description” Fragment Description

Besides it is important to note that the standard representation of the fragments allows, during the application of the whole SME process for creating agent design processes, an easy and quick selection from the repository for the assembly phase [16].

In [5] we illustrated how to document a fragment, here we give a brief hint on it, let us suppose to start from the representation of PASSI (see the previous subsection) and suppose that from the diagrams reported in Figures 5 and 7 we had identified the MMM element we want to be designed in the fragment, the related work product and then the portion of work to be done to have this result. What we have to do now is to model/represent this portion of work with its resulting product; in Figure 8 a SPEM workflow diagram is shown, here we can see the flow of work the specific role has to perform in order to produce the ”Role Description” diagram and all the inputs (obviously coming form other fragments) needed.

This latter diagram introduces a deeper level of detail with respect to the diagram shown in Figure 7, in fact it details the sequence of activity; it is helpful for the fragment documentation.

6 Discussions and Conclusions

The approach we use for adapting Situational Method Engineering to agent oriented design processes construction is composed of three phases, as it is in other approaches in literature [6][12]; in this paper we mainly consider the part regarding the (process) fragments documentation in a form that encourages the fragment extraction from an existing process and the repository construction.

We started this work from a set of existing design processes that we represented and fragmented in a standard fashion in order to extract a set of fragments coherent with the definition we gave.

The first problem we faced was to find a way for representing a design process coherently with the process metamodel we consider in our approach (activity, process role, work product and MAS metamodel element); we decided to follow a top-down approach for the design process representation using and extending, when necessary, the SPEM 2.0 specifications.

We decided on SPEM .0 and we think it is well suited to our purposes because it is an OMG standard, in our work we always had great attention for the standards, and above all because of the conceptual metamodel it is based on. The main elements we find in SPEM metamodel are: activity, process role and work product; due to their definition these elements well fit our definition of agent design process elements we gave in [5].

The breakdown structure, SPEM is composed of, let us to represent PASSI only using three kinds of diagrams; one of them is the extension we made on the work product dependency diagram provided by SPEM. This extension together with the other two ones we made was due to our concerns about MAS metamodel, whose elements cannot be represented used SPEM packages elements, and for representing the kind of work product we use in our work for categorize the fragments in the repository [16].

All the approaches present in literature draw on a more or less huge or formalized and structured repository of fragments and each researcher in the field of SME takes care of populating it by extracting fragments from existing design processes [1][7][13][14].

However guidelines on how to do that are still lacking; only in [12] a complete work on that can be found. There Ralyté proposes an approach based on the decomposition of existing processes into smaller components aiming at satisfying specific goals, but it seems that this approach is to bound to the rigidity of existing design processes (not originally created to be modular, as the same author says) and to the quality of the model provided for the process and the product parts of the design process.

Our approach is also based on the concept of decomposition but it tries to overcome these problems by firstly adopting a way for representing in the proper fashion a design process and then extracting from them fragments.

In the past we used SPEM 1.1 in our work and we modelled several agent design processes, we found several difficulties that are now over with SPEM 2.0; one of the greatest advantages we found in using SPEM with the proposed extension is that it is possible to represent a whole design process only through three diagrams from which all the essential information for fragmentation can be easily and quickly gathered.

Our research is principally focussed, and this was the starting point, on the application of Situational Method Engineering for the construction of ad-hoc agent oriented design processes but while working we realized that, due to the use of metamodel, to the elements we identified for a design process definition and to the use of SPEM for representing them, our approach results general enough to be applied to every kind of design process construction.

In the future we are going to experiment and to test SPEM 2.0 for other processes and to formalize a set of guidelines for correctly model an agent design process for the scopes of Situational Method Engineering applied to every application contexts and research fields.

References

1. Henderson-Sellers, B., Serour, M., McBride, T., Gonzalez-Perez, C., Dagher, L.: Process construction and customization. j-jucs 10(4), 326–358 (2004)
2. Brinkkemper, S.: Method engineering: engineering the information systems development methods and tools. Information and Software Technology 37(11) (1996)
3. Atkinson, C., Kuhne, T.: Model-driven development: A metamodeling foundation. IEEE Software 20(5), 36–41 (2003)
4. Cossentino, M.: From requirements to code with the PASSI methodology. In: Agent Oriented Methodologies, ch. IV, pp. 79–106. Idea Group Publishing, USA (2005)
5. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardisation to research. International Journal of Agent-Oriented Software Engineering (IJAOSE) 1(1), 91–121 (2007)
6. Gupta, D., Prakash, N.: Engineering Methods from Method Requirements Specifications. Requirements Engineering 6(3), 135–160 (2001)
7. Harmsen, A.F., Ernst, M., Twente, U.: Situational Method Engineering. Moret Ernst & Young Management Consultants (1997)
8. Henderson-Sellers, B.: Method engineering: Theory and practice. In: ISTA, pp. 13–23 (2006)
9. Kumar, K., Welke, R.J.: Methodology engineering: a proposal for situation-specific methodology construction. In: Challenges and Strategies for Research in Systems Development, pp. 257–269 (1992)
10. Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1), 58–78 (2006)
11. Software process engineering metamodel. Version 2.0. Final Adopted Specification ptc/07 03-03. Object management group (omg) (March 2007)
12. Ralyté, J.: Towards situational methods for information systems development: engineering reusable method chunks. In: Procs. 13th Int. Conf. on Information Systems Development. Advances in Theory, Practice and Education, pp. 271–282 (2004)
13. Ralytè, J., Rolland, C.: An approach for method reengineering. LNCS, pp. 27–30 (2001)
14. Brinkkemper, S., Saeki, M., Harmsen, F.: Assembly Techniques for Method Engineering. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, p. 381. Springer, Heidelberg (1998)
15. Shavrin, S.: Ontological multilevel modeling language. International Journal Information Theories & Applications 14 (2007)
16. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In: Proc. Of the Workshop on Objects and Agents, WOA 2006 (2006)
17. Seidita, V., Ralyté, J., Henderson-Sellers, B., Cossentino, M., Arni-Bloch, N.: A comparison of deontic matrices, maps and activity diagrams for the construction of situational methods. In: CAiSE 2007 Forum, Proceedings of the CAiSE 2007 Forum at the 19th International Conference on Advanced Information Systems Engineering, Trondheim, Norway, June 11-15, pp. 85–88 (2007)

Definition of Process Models for Agent-Based Development

Iván García-Magariño[1], Alma Gómez-Rodríguez[2], and Juan C. González-Moreno[2]

[1] D. Software Engineering and Artificial Intelligence
Facultad de Informática
Universidad Complutense de Madrid, Spain
ivan_gmg@fdi.ucm.es
http://grasia.fdi.ucm.es/

[2] D. de Informática (University of Vigo)
Ed. Politécnico, Campus As Lagoas,
Ourense E-32004 (SPAIN)
{alma,jcmoreno}@uvigo.es
http://gwai.ei.uvigo.es/

Abstract. As in other kinds of software development, the definition of process models in Multi-agent System (MAS) domain has many advantages. Some of these advantages are the better understanding which facilitates process measurement and improvement, and that the definition constitutes the basis for automating the process itself. The main goal of this paper is to provide a proper mechanism for defining agent-based development process models. For achieving this goal, an open-source editor tool and a technique for defining process models with the mentioned tool are presented. Both the editor tool and the technique provide MAS designers with a suitable mechanism for defining process models and are based on the Software Process Engineering Metamodel (SPEM). Although the main goal is to define process models for agent-based development, the proposed tool and technique can define any software process model, because they are based on a general-purpose software process metamodel. The utility of the tool and the technique is also justified, finally, by qualitative comparison with others.

Keywords: Multi-agent Systems, Development, Process, SPEM, Tool, Metamodel.

1 Introduction

Some authors [6,8], indicate that the modeling of processes for agent-based development is an increasing demand due to many reasons. Some of the Agent-oriented Software Engineering (AOSE) methodologies [4,7,9,19,20,23,24] may follow different process models depending on the system to be constructed. In these cases, the anticipated definition of the process, by means of its modelling, is useful for the right agent-based development. Besides, the process models can be shared among Multi-agent System (MAS) designers; in this way, the process models defined by experienced designers will be provided for helping inexperienced developers.

Moreover, defining the process models provides the basis for automating the process itself, in the same way as it is done in other engineering fields. This definition opens the possibility of customizing CASE tools for a particular process within a methodology.

This paper focuses on the definition of process models for agent-based development. This definition would be easier to do using an editor which allows designers to create the suitable diagrams and definitions in an automated manner. Several tools, such as [2,8,11], have been proposed, but they fail in satisfying all the necessities of the agent-based development or when describing the technique for defining the process using the editor. For this reason, this paper presents both an open-source editor tool and a technique for defining process models.

Although the main goal of the editor tool and mechanism is to define agent-based development processes, since they based on a general-purpose standard (SPEM [22]), they can be used in the definition of process models for any software development.

The structure of the remaining of the paper follows. Section 2 justifies the definition of a new tool, introducing the main drawbacks of some process model editor tools and making a qualitative comparison of the tools. Next section introduces the editor tool for modelling. Section 4 provides the technique for defining process models for agent-based development with the proposed tool. Finally, Section 5 introduces the conclusions and future work.

2 Comparison of Editor Tool with Others

The proposal of a new tool has to be justified, in order to indicate what are the advantages over others tools with the same or similar purpose. Although there is an increasing interest in process modelling and standardisation, encouraged by FIPA committee [10], there are still few tools which allow the definition of development processes. Among these tools, in this section we refer to APES [2], EPF [11] and Metameth [8] as the most relevant ones in the context previously exposed.

The approach for the comparison of the proposed editor with others can only be qualitative, because the work is in a stage where a deeper study will be needed for quantitative comparison. Nevertheless, it is authors' intention to address this objective comparison latter on.

APES, which current version is APES2, is software for process modelling. APES2 is, according to its authors, a process modelling software which follows the SPEM specification wrote by the OMG. The tool has several advantages: it is in conformity with a standard, enables the user to validate a process component, enables the user to easily modify a model and it is free (expensive and proprietary solutions can be avoided). In fact, APES has been split in four tools: a modelling tool (APES2), a presentation tool (POG), a publication tool (IEPP)

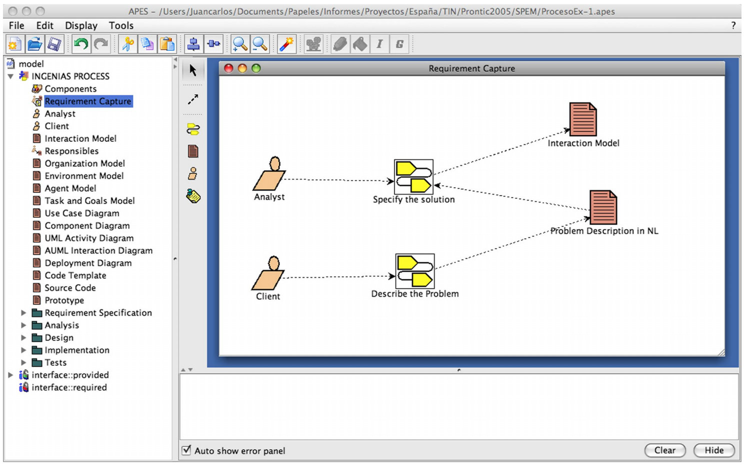


Fig. 1. Screenshot of APES tool when defining INGENIAS process

and an execution tool (PEACH). So, the comparison will focus on APES2 which addresses the same goal of the editor tool presented.

The authors tried initially to model INGENIAS process using existent tools, in particular APES2. In Figure 1, an attempt to model several activities of the INGENIAS process is shown. Initially, the guideline for the definition of process based on the use of the stable standard SPEM 1.1 was described with APES, trying to follow an approach which allows its adaptation to some of the proposals for SPEM 2.0. APES tool becomes not valid for this definition because it does not implement the full specification of SPEM. Moreover, it is not possible, for instance, to distinguish between packages and disciplines which is a fundamental point for the adopted guideline. In Figure 2, the definition of packages in APES can be seen, there is no disciplines option.

APES software provides support for some entities: Activity, Work, Work Product, Work Product State and Process Role. In what respect to diagrams it includes: Activity diagrams, Context diagrams, Work definition diagrams, Flow diagrams and Responsibility diagrams. As it can be easily seen, the tool does not cover completely SPEM standard. It lacks coverage for important features and entities, like discipline or guidance (these entities are covered by the tool presented in the paper). In addition, it has some limitations in what respects to verification or consistency checking.

On the other hand, the work done by the Process Framework Project (EPF) has two main goals: to provide an extensible framework for software process engineering and to provide exemplary and extensible process content for a range of software development. The first goal has been achieved by the creation of Eclipse Process Framework, which is a tool for defining a customizable software process engineering framework. A screenshot of the tool can be seen in Figure 3.

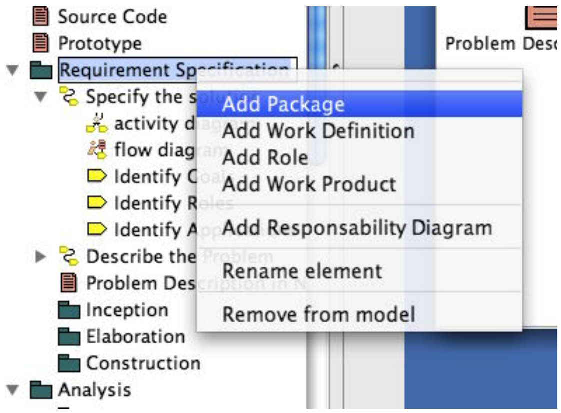


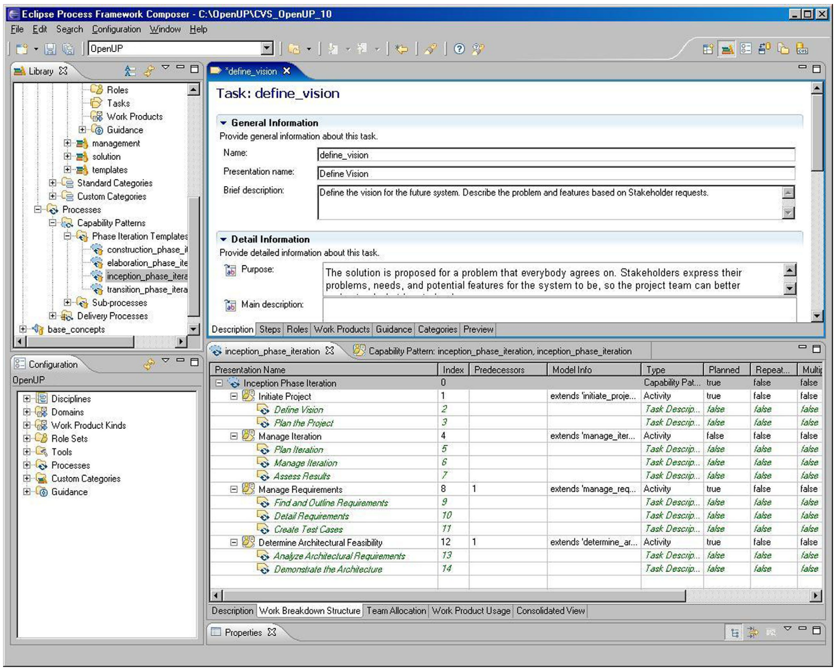
Fig. 2. APES tool has no option for Discipline

EPF is a tool with coverage of the fundamental entities and diagrams for process definition. The definition of this tool is based on XML schemas, in this sense is similar to the tool proposed in this paper but it differs in the structure of files generated. The editor tool proposed in this paper produces a XML document based in XMI standard for SPEM, while EPF does not follow XMI. Besides EPF generate several XML documents organized in a directory tree, so the inherent structure is reflected in the tree of documents. However the editor tool proposed in this paper provides just one structured standard XMI file (this means that the structure is enclosed in the hierarchical XML file).

The EPF tool follows SPEM in its (recently adopted) version 2.0. The tool proposed in this paper has been originally conceived following the version 1.1 of SPEM, but taking into consideration most of the novelties introduced in SPEM latest version. The tool extension for SPEM 2.0 has been done ad-hoc and provides more flexibility than EPF when modeling some concepts (such as optional relationships); because EPF is restricted by the rigid UML profile used in SPEM definition. This fact makes the proposed tool a real alternative to EPF when a more flexible definition of a concept is required.

An important characteristic of EPF is that it provides some integration with other software, for instance, it creates project schedules for Microsoft Project from project definition. However, the authors think that a more powerful integration is needed. For this reason, the final goal of the tool proposed is to use the XMI document generated as the basis for feeding a CASE tool which will provide validation and guide for software development. In this way the CASE tool will be adapted to the process of development defined for each particular system and will provide guides to user of what thing to do next, what the proposed scheduling for development will be, etc.

Recently a new tool for defining processes has been proposed. It is called Metameth [8] and has been constructed with a similar philosophy than the tool proposed in this paper; that is, following SPEM and using other tools for defining



Task: define_vision

General Information
Provide general information about this task.

Name: define_vision
Presentation name: Define Vision
Brief description: Define the vision for the future system. Describe the problem and features based on Stakeholder requests.

Detail Information
Provide detailed information about this task.

Purpose: The solution is proposed for a problem that everybody agrees on. Stakeholders express their problems, needs, and potential features for the system to be, so the project team can better

Main description:

Presentation Name	Index	Predecessors	Model Info	Type	Planned	Repeat...	Multi
Inception Phase Iteration	0			Capability Pat...	true	false	false
Initiate Project	1		extends 'initiate_proje...	Activity	true	false	false
Define Vision	*2*			*Task Descrip...*	*false*	*false*	*false*
Plan the Project	*3*			*Task Descrip...*	*false*	*false*	*false*
Manage Iteration	4		extends 'manage_iter...	Activity	false	false	false
Plan Iteration	*5*			*Task Descrip...*	*false*	*false*	*false*
Manage Iteration	*6*			*Task Descrip...*	*false*	*false*	*false*
Assess Results	*7*			*Task Descrip...*	*false*	*false*	*false*
Manage Requirements	8	1	extends 'manage_req...	Activity	true	false	false
Find and Outline Requirements	*9*			*Task Descrip...*	*false*	*false*	*false*
Detail Requirements	*10*			*Task Descrip...*	*false*	*false*	*false*
Create Test Cases	*11*			*Task Descrip...*	*false*	*false*	*false*
Determine Architectural Feasibility	12	1	extends 'determine_ar...	Activity	true	false	false
Analyze Architectural Requirements	*13*			*Task Descrip...*	*false*	*false*	*false*
Demonstrate the Architecture	*14*			*Task Descrip...*	*false*	*false*	*false*

Fig. 3. Screenshot of EPF

the process metamodel. The results are very interesting and the tool provides important functionalities for process definition. In this case, the tool seems to be useful and powerful, but there are few indications of how to use it for defining a new process. So the results of definition will relay on engineers experience.

3 The Process Model Editor Tool

From the MAS based process modelling point of view, the *FIPA Methodology Technical Committee* [10] has suggested the use of Software Process Engineering Metamodel (SPEM), due to *Object Management Group* (OMG) and introduced as a UML profile for software development process modelling. As it has been said before, there are other tools based on SPEM (i. e. APES [2] or EPF [11]) but they have some drawbacks. The tool presented in this paper tries to overcome some of those problems providing coverage to the whole SPEM specification. In this manner, the process model editor supports almost any process model, including the process models for agent-based developments.

The creation of a tool editor supporting the whole SPEM specification has extraordinary costs with the classical techniques; but using the Model-driven Development (MDD) [3], the costs can be affordable. In fact, the process model editor tool presented in this paper is built with MDD. In particular, the tool creation uses techniques from the *Domain-specific Model Languages* (DSML) [1] field. In first place, a language model (metamodel) is defined and, next, a framework generates the editor from the metamodel. In this way, the process of obtaining the editor is reduced to define properly the metamodel. This technique provides an efficient way of developing the editor and maintaining it.

At this moment, there are several metamodel languages available. Some of them are the MOF language [21], GOPRR [18,26] and Ecore, which is used by the *Eclipse Modelling Framework (EMF)* [5,12]. In order to decide among so many languages, two facts are important to be aware of. First, the SPEM specification [22] uses MOF language. Second, the Ecore language is used by the most reliable tools and frameworks. In particular, Ecore is used by EMF and EMF can generate an editor automatically from an Ecore metamodel. Therefore, we defined the SPEM metamodel with Ecore, from its definition in MOF, so that we got the SPEM editor generated automatically by EMF. In addition, this editor serialises their SPEM models using XMI documents, which are widely supported.

The models generated using the proposed editor represent software processes. The editor lets the user define these models with a graphical user interface. A more detailed view of the editor is available in [14]

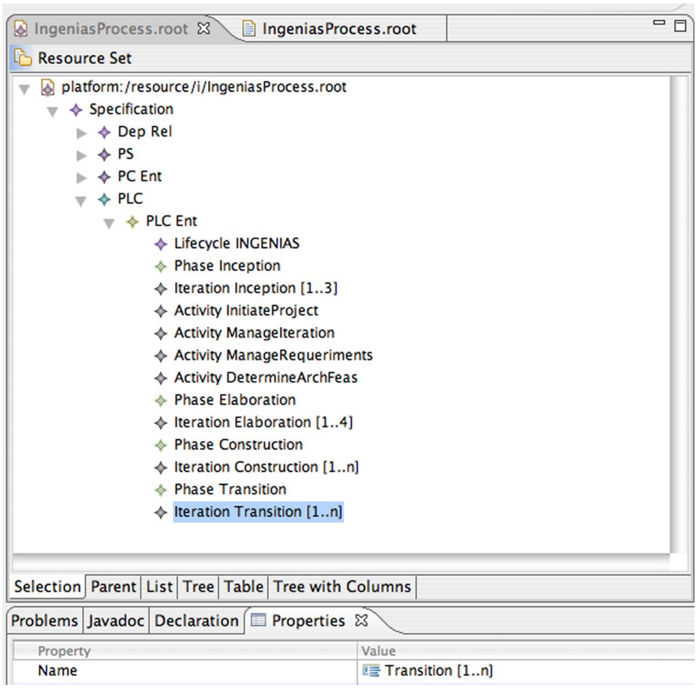
4 Technique for Defining Process Models

This section presents a technique for defining process models. The editor previously introduced has been used as tool during specification. The framework is based on the development of three orthogonal views of the systems: the view of lifecycle process, the disciplines used in the process view and the guidance and suggestions view. The first view proposed describes the path which must be followed for obtaining as final product a MAS jointly with the activities that must be accomplished along the path. The second view establishes the disciplines of the process, the roles and participants and the tasks to perform, which will consume and produce a set of products. Finally, the guidance and suggestions view details products, works and roles which constitute model elements of the other views.

The case study selected for introducing this model guide is the Unified Development Process of INGENIAS [15,16]. The process is defined as the integration of INGENIAS metamodels and the Unified Development Process (UDP) [17] in what refers to analysis and design disciplines. UDP distributes the tasks of analysis and design in three consecutive phases: *Inception*, *Elaboration* and *Construction*, with several iterations (where iteration means a complete cycle of development, which includes the performance of some analysis, design, implementation and proof tasks). The sequence of iterations leads to the procurement of the final system.

Several steps must be followed in the definition of a process model for agent-based development. These steps are the following ones.

1. Identify the process model with an existent process model, if possible. Before defining a new process model, it is useful to have in mind the existent process models. In [6] some of the most known process models are matched with



```
platform:/resource/i/IngeniasProcess.root
  Specification
    Dep Rel
    PS
    PC Ent
    PLC
      PLC Ent
        Lifecycle INGENIAS
        Phase Inception
        Iteration Inception [1..3]
        Activity InitiateProject
        Activity ManageIteration
        Activity ManageRequeriments
        Activity DetermineArchFeas
        Phase Elaboration
        Iteration Elaboration [1..4]
        Phase Construction
        Iteration Construction [1..n]
        Phase Transition
        Iteration Transition [1..n]
```

Property	Value
Name	Transition [1..n]

Fig. 4. A possible INGENIAS Lifecycle defined with the Editor

some AOSE methodologies. The matching between methodology and process must be a good point of departure for starting the definition. Nevertheless, it is worth mentioning that a methodology may use several process models depending on the MAS.

2. Define the lifecycle view.
3. Define the disciplines view.
4. Define the guidance and suggestion view.

All the aforementioned technique steps are further explained in the following subsections.

4.1 Lifecycle View

The specification starts by modeling the particular lifecycle wanted for the development. This specification is done defining the phases, activities and iterations required to complete the development, and delaying the modeling of participants and products-to-obtain to a latter step. The development path is divided in consecutive phases and each of them in a different number of iterations. The number of iterations is defined adding on the iteration's name the maximum and minimum number of allowed repetitions (for instance [1..3]). That will work if the

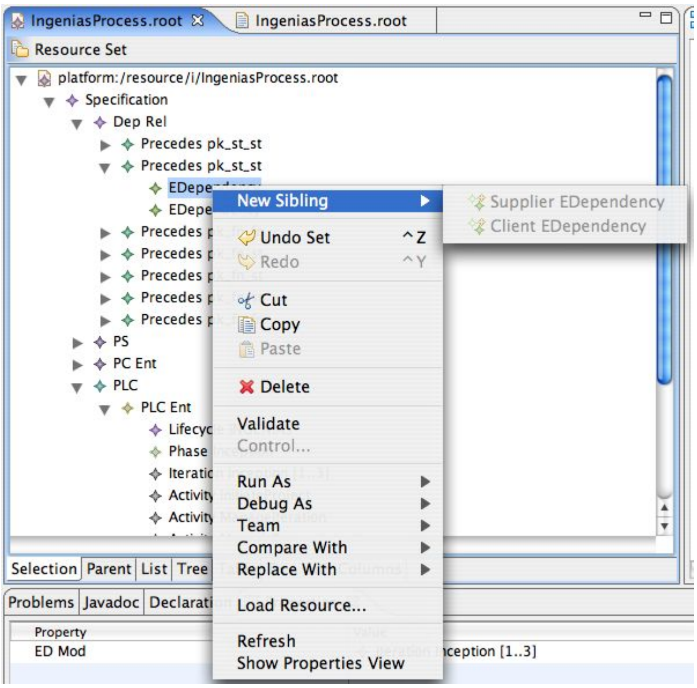


Fig. 5. Definition of phases precedence relationship using the editor

number is always the same, in other case, it is mandatory to itemize each of the iterations. This definition constitutes one of the examples of the flexible ad-hoc definition introduced when comparing the tool with EPF in section 2. Figure 4 shows iterations definition for all the phases of INGENIAS process. Each phase has an `Iteration` element which includes the minimum and maximum number of suitable iterations. In other case, if the number of iterations is optional, for instance if there can be none or five iterations, then the name of the iteration would end in (5).

Using the editor described in previous section, the representation of the iterative and incremental lifecycle is made in several steps. In first place, the elements (phase, iteration and activities) which constitute the lifecycle are defined. Initially, a child of `Specification` of kind `PLCProcess Lifecycle` must be created. After, the analyst must define an entity of `Lifecycle` type, four more entities, corresponding to each phase, of `Phase` type and one entity of `Iteration` type, which will include in its name, as it was said before, the minimum and maximum number of allowed iterations for each phase. The following step is the definition of the existing temporal and membership relationships among these elements. Figure 4 presents a snapshot of the editor when doing the iteration definition for a phase of lifecycle for a possible INGENIAS process model. Figure 5, on the other hand, shows how the precedence relationships among lifecycle, phases and iterations are defined creating the children entities `Dep Rel` of type `Precedes`.

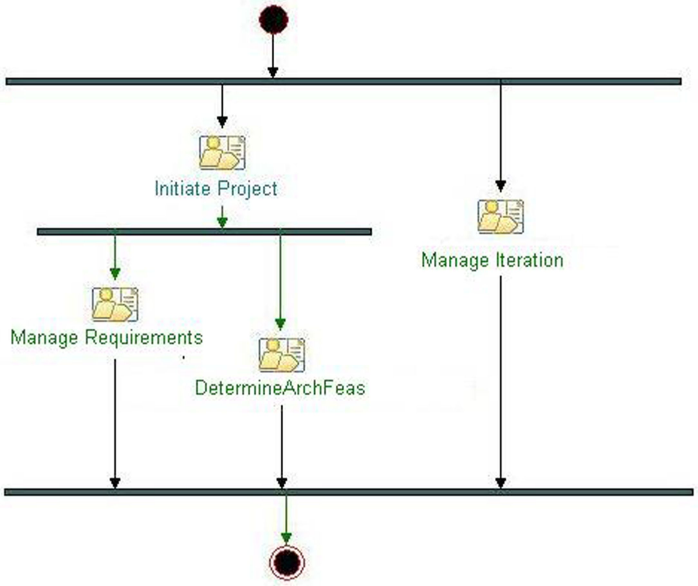


Fig. 6. Definition of phases precedence relationship using the editor

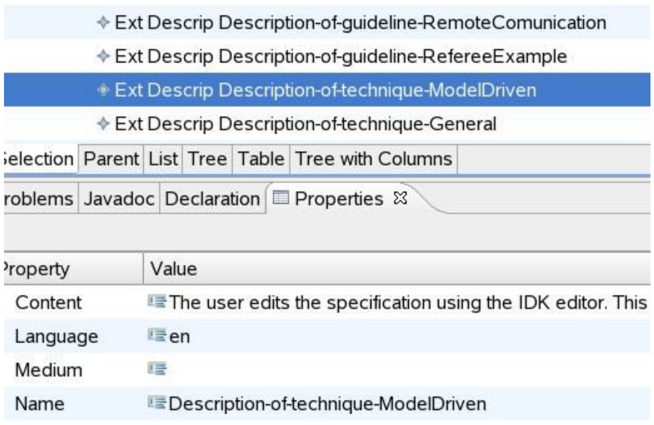
Once the previous activities are done, it is necessary to define for each iteration the phase it belongs to, and what are the activities that must be performed. For instance, Inception phase in the described process implies the accomplishment of several activities shown in Figure 6 (diagram obtained using EPF).

Next step, after having specified the development process along with the activities to perform in each iteration, is defining the disciplines implied in this activities formalisation.

4.2 Disciplines View

Disciplines in SPEM determine process packages which take part in the process activities, related with a common *subject*. That is, disciplines represent a specialisation of selected subactivities, where these new subactivities can not appear in other packages or disciplines. In the case study, the disciplines which are included in the Development Process are *RequirementsSpecification*, *Analysis*, *Design* and *Implementation*. The definition of these disciplines is made by creating inside of `Specification` a child of type `PC Ent` (*Process Component* entities). Next, the four disciplines are included inside by selecting new entities of `Discipline` type.

In the next step, the methodology suggests that, before detailing the subactivities (*tasks*) which constitute each discipline, the analyst indicates the participants. This definition is made using as basis the different roles that each participant will play along the project and the products used. In the selected case study the roles implied are: *Analyst*, *Architect*, *Developer*, *Tester*, *Project*



Ext Descrip Description-of-guideline-RemoteComunication
Ext Descrip Description-of-guideline-RefereeExample
Ext Descrip Description-of-technique-ModelDriven
Ext Descrip Description-of-technique-General

roperty	Value
Content	The user edits the specification using the IDK editor. This
Language	en
Medium	
Name	Description-of-technique-ModelDriven

Fig. 7. Guidances External Descriptions. The attributes (content, language, medium and name) of the selected External Description are shown in the *Properties* tab. In this case, the reader can see the beginning of the external description of the *Model-Driven* technique.

Manager and *FreeMan* and the products used are the following ones: *Use Case Diagram*, *Environment Model*, *Organisation Model*, *Interaction Model*, *Agent Model*, *Object and Task Model*, *Glossary*, *NonFunctional Requirement* and *Stakeholder Vision*.

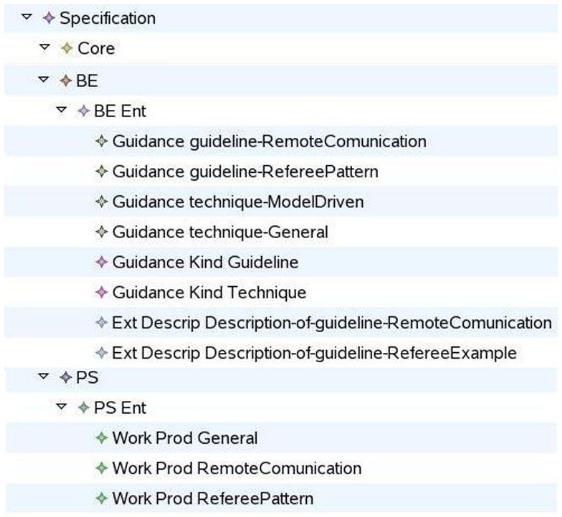
In the editor, the mechanism for including roles and products is to create, in `PS Ent`, entities with types `Proc Rol` and `Work Prod` respectively.

Once the participants, products, disciplines and process are defined, each activity must be divided for each iteration depending on the task pertaining to each discipline. The activities must be performed by any of the specified participants consuming and/or producing any of the existent products. The process for making up this step is similar to the previous ones, but modelling tasks like `Step` inside `PS Ent` and relations between activities and task like a `PS Rel` of type `RAc Step`. The relationships of making an activity by a participant will be defined using relations `PS Rel` of type `RAss`. In version 1.1, activities and consumed and/or produced products have the limitation that they can not be related directly; so this relationship is established using as intermediates the discipline where it is used and the role which manages it.

4.3 Guidances View

The *Guidances* provide information about certain model elements. Each *Guidance* is associated to a *Guidance Kind*. SPEM provides a basic repertoire of Guidance Kinds, but it is flexible about Guidance Kinds.

In the editor (Figure 8), the `Guidance` and `GuidanceKind` elements can be added to the `BE Ent` element (entities of the *Basic Elements* SPEM package). Each Guidance must be associated with the `RGuidK` relationship in the `BE Rel`



```
Specification
  Core
  BE
    BE Ent
      Guidance guideline-RemoteComunication
      Guidance guideline-RefereePattern
      Guidance technique-ModelDriven
      Guidance technique-General
      Guidance Kind Guideline
      Guidance Kind Technique
      Ext Descrip Description-of-guideline-RemoteComunication
      Ext Descrip Description-of-guideline-RefereeExample
  PS
    PS Ent
      Work Prod General
      Work Prod RemoteComunication
      Work Prod RefereePattern
```

Fig. 8. Entities and Relationships related to the Guidances. Entities are Guidances, Guidance Kinds, External Descriptions and Work Products. Relationships link these entities.

element. Each Guidance must be, also, associated with a `Model Element`, using the relationship `RAnnot` in the `BE Rel` element. In this example (Figure 8), all the guidances are linked with work products which are model elements. Each guidance (guideline or technique) is linked to an *External Description*(element `Ext Descrip`), which contains a complete explanation of the guidance. The guidance external description has an attribute named `content` with the guidance content (see Figure 7). It also contains an attribute indicating the language; in the example, English is selected. The `Ext Descrip` element can be added to the `BE Ent` element to define an external description. The guidance can connect with its external description using the `RPresentation` element in the `Core Rel` element (relationships of the *Core* SPEM package).

In multi-agent systems, we recommend specially to use two kinds of guidance: *Technique* and *Guideline*. The *technique* provides an algorithm to create a work product. The *guideline* is a set of rules and recommendations about a work product organisation.

As examples, the most relevant INGENIAS Guidances are described below, and the reader can see them on Figure 8.

– *General Technique*. It indicates how to develop a complete INGENIAS specification, so that the multi-agent system, generated from the specification, runs itself. This technique gives the algorithm for developing a *General* work product. The general product is supposed to be any kind of product.
– *Model-Driven Technique*. This guideline recommends the user to follow the *Model-Driven Development* [3] principles. Pavon [25] explains explicitly how

to develop an INGENIAS MAS driven by the model. Basically, the user edits the specification using the IDK editor. This specification is the model on which the development is based. The code generator obtains the Java code for JADE platform from the specification. In the specification, the user can add some pieces of code. This concrete piece of code remains on the specification so that, the next time the system is generated, the code is not overwritten. In addition, working only on the specification (model) fastens the software process. This technique is recommended for all the work products, represented by the *General* work product.
- *Remote Communication Guideline.* It provides rules and recommendations to organise a work product focused on remote communication. The INGENIAS 2.6 distribution provides an example, named *IAF-cinema*, that follows this guideline. This guideline is applied only to the work product named *RemoteCommunication.*
- *Referee Pattern Guideline.* It provides a *referee-pattern* INGENIAS MAS organisation. An agent plays the role of a referee. This agent is in charge of controlling the communication among the other agents and stopping the multi-agent system when necessary. The *delphi* multi-agent system [13] follows this guideline.

5 Conclusions and Future Work

The definition of the software process engineering model of a MAS methodology makes it easier to learn and use a MAS methodology, in several manners. This paper presents a technique and an editor tool (based on SPEM and DSML techniques) which allow the definition of process models for agent-based development. In particular, the technique and the proposed editor are used for the definition of the Unified Development Process for INGENIAS methodology. The application of the tool to a particular process definition proves its utility for the objective it was created for.

Furthermore, this work can guide a MAS developer through the steps of definition of a development process for MAS construction. The description of the steps to follow provided in the paper, can simplify the definition of processes for non expert engineers.

In addition, the paper addresses a comparison between the editor proposed by authors and similar available tools. As conclusion of this comparison, it is thought that the proposed tool has all the functionality needed for defining process models for agent-based development.

Next step in tool evolution will be to integrate the process editor with a tool for methodology support, so each software process engineering model will be available in the methodology support tool. In this way, the tool for development will be able to guide the user through the development steps, using the information provided by the XMI documents generated by the tool proposed in this paper. For instance, the defined INGENIAS software process described in the paper can be integrated with the INGENIAS Development Kit (IDK) tool.

In the future, several software process engineering models of several MAS methodologies can be defined with the presented editor. These process models can assist the MAS designer in selecting the appropriate methodology and process model for a specific MAS.

Acknowledgements. This work has been supported by the following projects: *Methods and tools for agent-based modelling* supported by Spanish Council for Science and Technology with grant TIN2005-08501-C03-01; *Methods and tools for agent-based modelling* supported by Spanish Council for Science and Technology with grant TIN2005-08501-C03-03 and Grant for Research Group 910494 by the Region of Madrid (Comunidad de Madrid) and the Universidad Complutense Madrid.

References

1. Amyot, D., Farah, H., Roy, J.: Evaluation of Development Tools for Domain-Specific Modeling Languages. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 183–197. Springer, Heidelberg (2006)
2. APES2: A Process Engineering Software, `http://apes2.berlios.de/en/links.html`
3. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. Software, IEEE 20(5), 36–41 (2003)
4. Bernon, C., Cossentino, M., Pavón, J.: Agent-oriented software engineering. Knowl. Eng. Rev. 20(2), 99–116 (2005)
5. Budinsky, F.: Eclipse Modelling Framework: Developer's Guide. Addison Wesley, Reading (2003)
6. Cernuzzi, L., Cossentino, M., Zambonelli, F.: Process models for agent-based development. Engineering Applications of Artificial Intelligence 18(2), 205–222 (2005)
7. Chella, A., Cossentino, M., Sabatucci, L., Seidita, V.: Agile PASSI: An Agile Process for Designing Agents. International Journal of Computer Systems Science & Engineering. Special issue on Software Engineering for Multi-Agent Systems (May 2006)
8. Cossentino, M., Sabatucci, L., Seidita, V., Gaglio, S.: An Agent Oriented Tool for New Design Processes. In: Proceedings of the Fourth European Workshop on Multi-Agent Systems (2006)
9. Cuesta, P., Gómez, A., González, J., Rodríguez, F.J.: The MESMA methodology for agent-oriented software engineering. In: Proceedings of First International Workshop on Practical Applications of Agents and Multiagent Systems (IWPAAMS 2002), pp. 87–98 (2002)
10. Consiglio Nazionale delle Ricerche (CNR): Activity of the FIPA Methodology Technical Committee
11. Eclipse: Eclipse Process Framework (EPF), `http://www.eclipse.org/epf/`
12. Moore, B., et al.: Eclipse Development using Graphical Editing Framework and the Eclipse Modelling Framework. IBM Redbooks (2004)
13. García-Magariño, I., Agera, J.R.P., Gómez-Sanz, J.J.: Reaching consensus in a multi-agent system. In: International workshop on practical applications on agents and multi-agent systems, Salamanca, Spain (2007)

14. García-Magariño, I., Gómez-Rodríguez, A., González, J.C.: Modelando el Proceso de Desarrollo de INGENIAS con EMF. In: 6th International Workshop on Practical Applications on Agents and Multi-agent Systems, IWPAAMS 2007, Salamanca, Spain, November 12-13, pp. 369–378 (2007)
15. Gómez-Sanz, J.J.: Modelado de Sistemas Multi-Agente. Ph.D thesis, Departamento de Sistemas Informáticos y Programación, Universidad Complutense Madrid (2002)
16. Gómez-Sanz, J.J., Fuentes, R.: Agent oriented software engineering with ingenias. In: Fourth Iberoamerican Workshop on Multi-Agent Systems (Iberagents 2002), a workshop of IBERAMIA 2002, the VIII Iberoamerican Conference on Artificial Intelligence (2002)
17. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
18. Kelly, S.: GOPRR Description. Ph.D dissertation, Appendix 1 (1997)
19. Mas, A.: Agentes Software y Sistemas Multi-Agentes. Pearson Prentice Hall (2004)
20. O'Malley, S.A., DeLoach, S.A.: Determining when to use an agent-oriented software engineering pradigm. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 188–205. Springer, Heidelberg (2002)
21. OMG. Meta Object Facility (MOF) Specification
22. OMG. Software Process Engineering Metamodel Specification (2005)
23. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In: Proceedings of the Third International Workshop on Agent Oriented Software Engineering, at AAMAS (2002)
24. Pavón, J., Gómez-Sanz, J.: Agent Oriented Software Engineering with INGENIAS. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS, vol. 2691, pp. 394–403. Springer, Heidelberg (2003)
25. Pavón, J., Gómez-Sanz, J., Fuentes, R.: Model Driven Development of Multi-Agent Systems. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 284–298. Springer, Heidelberg (2006)
26. Tolvanen, J.-P.: GOPRR metamodeling language (2000)

Methodology Fragments Definition in SPEM for Designing Adaptive Methodology: A First Step

Sylvain Rougemaille[1], Frederic Migeon[1], Thierry Millan[2], and Marie-Pierre Gleizes[1]

[1] SMAC team,
[2] MACAO team,
IRIT Computer Science Research Institut of Toulouse,
Université Paul Sabatier,
118 Route de Narbonne,
F-31062 TOULOUSE CEDEX 9, France
{rougemai,migeon,Thierry.Milan,Marie-Pierre.Gleizes}@irit.fr
http://www.irit.fr

Abstract. The aim of this paper is to highlight how SPEM (Software and System Process Engineering Meta-model) 2.0 OMG (Object Management Group) can participate to design adaptive methodology process. The idea follows the FIPA Methodology Technical Committee (TC) one which consists in expressing a methodology in several fragments. Then, designer has to combine the relevant fragments to compose his own methodology. In this paper, we have chosen SPEM 2.0 OMG to express the fragments. The latest SPEM version improves methodology content and process re-usability, by introducing new capabilities as a clear separation between structural and dynamic methodology concerns. Those improvements in the field of methodology specification, are studied to determine their interests in the scope of Agent-Oriented Software Engineering (AOSE) and particularly, their impact on "methodology fragments" definition. ADELFE and PASSI methodologies have been taken as example to illustrate the use of SPEM 2.0 in the scope of "fragment" definition. In this paper, only the first step of the general objective consisting in expressing the fragments, is done and presented.

Keywords: SPEM 2.0, ADELFE, Methodology Fragments, Agent Oriented Software Engineering, Process Engineering.

1 Introduction

Many agent-oriented methodologies have been developed last decade (e.g. [1,12]: ASPECS [7], ADELFE [2], Gaia [22], INGENIAS [9], PASSI [6], Prometheus [18], SODA [17], Tropos [4]). Each has its own specificities: ADELFE is dedicated to adaptive system and cooperative agents design, ASPECS is dedicated to holonic multi-agent systems, Gaia focuses on static organization and roles, whereas

PASSI focuses on agent social aspects thanks to ontology, SODA highlights the notion of environment, etc. However, Agent-Oriented Software Engineering (AOSE) research community agrees the necessity of several methodologies and advocates the impossibility to build one general and universal one as it was pointed out in works like [10]. Thus, we cannot assume these methodologies can be applied to build every multi-agent applications. It seems that some methodologies or some parts of methodologies are more relevant than other to achieve some kinds of task. For example Tropos treats very well the preliminary requirements and provides models to realize it. That is the reason why the FIPA Methodology TC has proposed to define fragments. A fragment represents a portion of a methodology. Then, as it is also explained in [11], designers can choose methodological components from different methodologies in order to build their own relevant one. Software and System Process Engineering Meta-model (SPEM) is a standard specified by the Object Management Group (OMG) which latest revision 2.0 has just been adopted [16]. Its scope is the definition of a minimal set of concepts to design and manage software and system development process. In this paper, our aim is to highlight the relevance of this meta-model to express FIPA fragments.

Section 2 briefly describes the ADELFE methodology process which is used to illustrate the concepts developed in the paper. Section 3 presents the SPEM 2.0 OMG freshly adopted standard and studies more specifically new concepts such as *Method Plugin*, that are promising in the scope of AOSE. We also argue that this latest OMG vision of software process modeling is compliant to the FIPA Methodology TC concepts of “Methodology fragment” [8] (see section 5). Section 4 focuses on the aspects of SPEM 2.0 which are interesting for AOSE. Section 5 defines the fragment notion and the translation between the fragment and SPEM 2.0; this is illustrated on ADELFE and PASSI in section 6. The papers ends with the analysis and some perspectives of this work.

2 Introducing ADELFE Methodology

In order to illustrate the use of SPEM 2.0 [16] in AOSE, the ADELFE methodology [2] which was described with the previous SPEM version, has been taken as an example throughout the paper. ADELFE is devoted to the development of softwares with emergent functionalities and conforming to the Adaptive Multi-Agent System (AMAS) theory [3]. It is based on the Rational Unified Process (RUP) which was modified to fit specific AMAS needs. We recently have migrated this definition in SPEM 2.0 thanks to the Eclipse Process Framework (EPF)[1]. The following sections are illustrated with this definition conforming to the SPEM 2.0 recommended notations (SPEM 2.0 profile). ADELFE consists of five phases:

– *Preliminary* and *Final requirements*: they represent typical phases in object-oriented software development and are based on the RUP. However,

[1] `http://www.eclipse.org/epf`

in addition to the classical approach, they define AMAS specific tasks such as precise study of the system environment.
- *Analysis*: this is a specific phase that allows analysts to determine whether an AMAS is needed or not.
- *Design*: this phase is devoted to the determination of software architecture. It strongly depends on the previous ones (object or AMAS specific design).
- *Development*: this is a model-driven phase which allows automatic code generation of the previously designed agents. This phase is still under development and was presented in [19].

This paper focuses on the analysis phase. It is composed of four tasks devoted to the definition of a primary software architecture from the requirements that have been previously established (see figure 1). It determines the adequacy between the problem domain and an AMAS solution. This is done thanks to the "Verify AMAS Adequacy" task that allows the "Agent Analyst" to elect an AMAS approach, if needed, by answering questions about the systems functionalities.

To ease the analyst task, a visual tool which provides the adequacy degree at local and global level, has been developed. This verification task is specific to the ADELFE methodology, thus it has been defined as a fragment, and is a part of section 6 examples.

3 SPEM 2.0 Overview

SPEM 2.0 is a MOF 2.0 [13] based meta-model that defines extension to the UML 2.0 infrastructure [14] meta-model. A notation has also been defined, which is based on a UML 2.0 superstructure [15] profile. The following use this notation to illustrate the underlying meta-model. Roughly, the SPEM 2.0 meta-model consists of seven packages:

- The *Core* package defines SPEM 2.0 foundations: its main concepts and the way they can be extended (based on UML 2.0 infrastructure)
- The *Process Structure* package contains the basics elements to describe a development process as a breakdown structure.
- The *Process Behavior* package contains concepts enabling the description of process execution (*State,Transition, ControlFlow*, etc.). It does not define a specific formalism but links process structure and external behavioral concepts letting implementers feel free to select the appropriate language (UML activity or state machine diagrams, BPMN[2], etc.)
- The *Managed Content* package defines concepts for textual descriptions of processes and methodology contents.
- The *Method Content* package defines the core concepts of every methodology such as *Tasks*, *Roles* and *Work products.*
- The *Process With Method* package binds method contents (i.e. what have to be done) to process structure elements (i.e. the scheduling of these tasks).

[2] Business Process Modeling Notation `http://www.bpmn.org/`

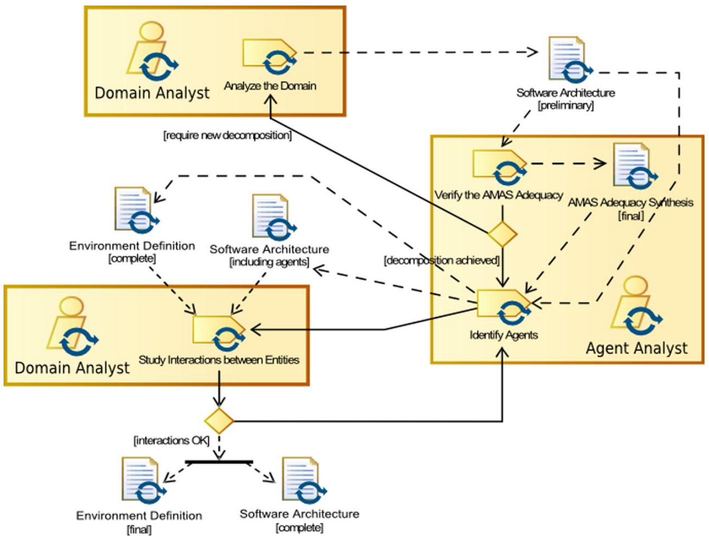

Domain Analyst
Analyze the Domain
Software Architecture [preliminary]
[require new decomposition]
Verify the AMAS Adequacy
AMAS Adequacy Synthesis [final]
Environment Definition [complete]
Software Architecture [including agents]
[decomposition achieved]
Identify Agents
Agent Analyst
Domain Analyst
Study Interactions between Entities
[interactions OK]
Environment Definition [final]
Software Architecture [complete]

Fig. 1. ADELFE analysis phase workflow

– The *Method Plugin* package defines large scale plugin mechanisms to ease the definition of separate extensible methodology. It provides the concept of *Method Library* which is the container for *Method Plugin* and *Configuration* (those two concepts and their interests are discussed in further sections).

As a process engineering meta-model, SPEM 2.0 allows to model, document, present, manage, interchange development processes and methods. *Phases* can be described from a static or structural point of view, in terms of *Method Content* concepts, or from a more dynamic point of view, using *Process with Methods* concepts. Figure 1 is an example of a workflow (UML activity diagram) describing the tasks sequence and the products involved in the ADELFE analysis phase.

4 SPEM 2.0 Capabilities

SPEM 2.0 structure has been improved and some of its concepts have been clarified, for instance, a clear separation have been defined between *Method Content Element* and *process Element.* Those improvements bring capabilities that we found interesting in the scope of agent oriented methodology fragments definition and use.

4.1 Concerns Separation

SPEM 2.0 provides clear separation between the definition of methodology contents *Tasks, Work Products* and *Roles* and their application in a specific delivery process (Activities, Role Use, Work Product use). In fact, those two concerns are respectively managed by the *Method Content* and *Process Structure* SPEM packages. Thus, process description is separated from the structure, it could be defined on its own (*Process Behavior* package) as, for example, an extension of UML Superstructure behavioral concepts (*Activity* and *State Chart* diagrams specialization). This separation is enforced by the definition of different *PackageableElements*: on one hand *MethodContentPackage* and on the other *ProcessPackage* which represents two distinct "containers". The first one contains the building bricks of methodologies (*Task, WorkProduct, Guidelines,...*), the second defines their use through *BreakdownElements* such as *Activity, TaskUse, RoleUse, WorkProductUse* and so on.

4.2 Modularity and Re-usability

At the Method level. The *Method Plugin* package authorizes modularization, eases extensibility and thus ensures that every methodology could be tailored in the more suitable way. SPEM 2.0 allows the description of *Method Plugin* which can be considered as a kind of extensible method packages container; it defines a whole method (content and use). The idea can be summarize as follow, a *Method Library* defines both *Method Plugins* and *Method Configurations* that represent respectively the content of the library and visibility "rules" over this content; in other words, building bricks that method engineers can select (the set of methods contained by the library), and what is effectively presented to the end-user (parts of those methods covering specific needs of the process). *Method Plugin* defines a first level of modularity. In addition, *Method Library* concept represents the topmost container for plugins and configurations, this last notion constitutes a kind of repository.

At the Process Level. *ProcessPattern* is a special kind of activity defined in the SPEM 2.0 base plugin, such as *Iteration, Phase,* etc. It refers to an un-applied general process, i.e. a breakdown of elements unbound to *Method Content* elements. It could be used as a building brick for development life cycle, the rationale for this concept is to describe reusable cluster of activities known to achieve the production of some deliverables (sets of Work products) in an efficient way.

Within a Process. Considering a fine-grain level of modularity and encapsulation, SPEM 2.0 offers the concept of *Process Component*, which conforms to the notion of software component [21] (provided and required interfaces, white or black box vision, ports, assembly, etc.). Taking a closer look to the adopted specification[16], it is a specialization of *ProcessPackage*, which is the primary process element container (*Activities, Task Use, Work Product Use*, etc.). It defines a process within an activity as a "black box" simply qualified by *Work Product Ports* that could gather its inputs and outputs (as could be software components interfaces). The main goal of process component is to keep some part of a process unresolved, i.e. the development phase, in order to choose the better implementation, using a code-centric approach or a model-driven one for example.

5 AOSE Methodology Fragments

This section presents the notion of Methodology Fragment in the scope of AOSE. It is promoted by the FIPA Methodology TC [8] and shares principles with *Situational Method Engineering* as it was quoted in [20]. FIPA method fragments are tightly connected to the SPEM 1.0 OMG standard as they use the same main concepts: *Activities, Artifacts, Role* and so on.

5.1 Definition

This section presents the method fragment as it has been defined by the FIPA methodology TC, an much more extensive definition of this proposition can be found in [5]. According to the FIPA, a fragment is composed of the following parts:

1. A portion of process.
2. The result of the work. It represents some kind of product/artifact like (A)UML/UML diagrams, text documents, etc.
3. Some preconditions, kind of constraints specifying when it is possible to start the process specified in the fragment because of missing required inputs or because of guard condition violation.
4. A list of concepts (related to the MAS meta-model) to be defined (designed) or refined during the specified process fragment.
5. Guideline(s) that illustrates how to apply the fragment and best practices related to that.
6. A glossary of terms used in the fragment (to avoid misunderstandings and to ease re-usability).
7. Composition guidelines - A description of the context/problem that is behind the methodology from which the specific fragment is extracted.
8. Aspects of fragment. Textual description of specific issues like for example: platform to be used, application area, etc.
9. Dependency relationships useful to define fragments assembly.

It should be noticed that not all of these parts are always mandatory. However, from this definition, it is already possible to note some interesting points. First of all, fragments use concepts that are still defined in the SPEM 2.0 specification (*Activities, Work Products, Roles*) but it needs clarification, because concepts have been moved into separate packages and are associated thanks to new elements such as *WorkProductUse* and *RoleUse.*

5.2 Fragment Compliance with SPEM 2.0

Rationale. SPEM possesses a wide audience and its use is enabled by many implementations. We advocate that being compliant with this “de facto” standard is important to broaden the use of agent-oriented methodologies and principles. Furthermore, as fragments aim to provide a common framework (language, repository) for agent-oriented methodologies and that the latest SPEM version partially meets these requirements we propose a translation, mapping between FIPA fragments and SPEM.

Mapping Fragments to SPEM 2.0. A method fragment is a portion of a development process defining deliverables (work products), guidelines, preconditions, and sets of concepts and key-words characterizing it in the scope of the method in which it was defined. Considering fragments through this rough definition, eases the determination of the more suitable SPEM 2.0 concepts, if any. However, one of the proposals of [5], which presents an enhanced version of the fragment meta-model from the FIPA methodology TC, is that fragments should be considered from different points of view whether you are interested in their reuse, storing, implementation or in the definition of the process they represent. So it seems obvious that the election of a method fragment depend on the point of view and the requirements defined by the method engineer. By taking a closer look to the SPEM 2.0 specification, *Method plugin* package provides concepts that fulfill most of FIPA requirements:

- *Method Plugin* defines by the means of *Method content packages* and *Process packages* sets of reusable process portions (see section 5.1). It goes further than this by splitting those reusable parts into methodology contents and process.
- FIPA vision of fragment corresponds to a process portion. Of course, this portion must embody some composition rules or constraints. FIPA elected *Process Component* as the SPEM 1.0 closer concept for fragment [5]. This notion has been improved and clarify in the latest specification so that it seems it better fits fragment needs. In fact, it defines precise input and output constraints in terms of *Work Product Ports*. A process component encapsulates a single Activity which can be broken down into sub-activities representing the process leading to be delivered of output work product using the declared inputs.
- Referring to the above description, fragment is equipped with glossary, guidelines and other textual descriptions or concepts intended to ease method engineer to achieve its composition. All those elements can be seen as *Guidance* special kinds. In the *Core Package Guidance* is an *Extensible Element* specialization, therefore it can be extended and defines its own *Kinds. Glossary, Aspect, Guideline* and *Composition Guideline* can be derived from *Guidance* and applied to any elements defined in the *Process Component.*
- In [20] authors promote the idea of a common repository for method fragments indexed thanks to concepts defined by MAS meta-models, they also have implemented it. SPEM integrates the concept of repository: *Method Library* which is the container for *Method Plugins.* A library contains several plugins, both *Method Content* and *Process Package* elements of these plugins can be referred by *Method Configuration* to tailor and to present a new Software Engineering Process (SEP).
- Dependency relationship are defined at different levels: between plugins, between process component as well as mapping from process pattern and the methodology contents they use. In fact, it only depends on fragment granularity.

Granularity. Fragment does not match a single SPEM 2.0 concept but should be considered as different ones depending, for instance, on its granularity. As SPEM offers different re-usable concepts with different granularity, from *task* to *Method Plugin.*

Concerns. Fragment is a portion of process. However, process element in SPEM 2.0 has been divided into definition and use. Thus, this separation needs to be considered while mapping fragment to SPEM concepts. Fragments that define elementary work part, related products and roles should be mapped to *Method Content* concepts. Whereas fragments defining some "good practice" or a common way to deliver a kind of products (i.e. an Model Driven approach to produce code), should be mapped to *Process Content* concepts.

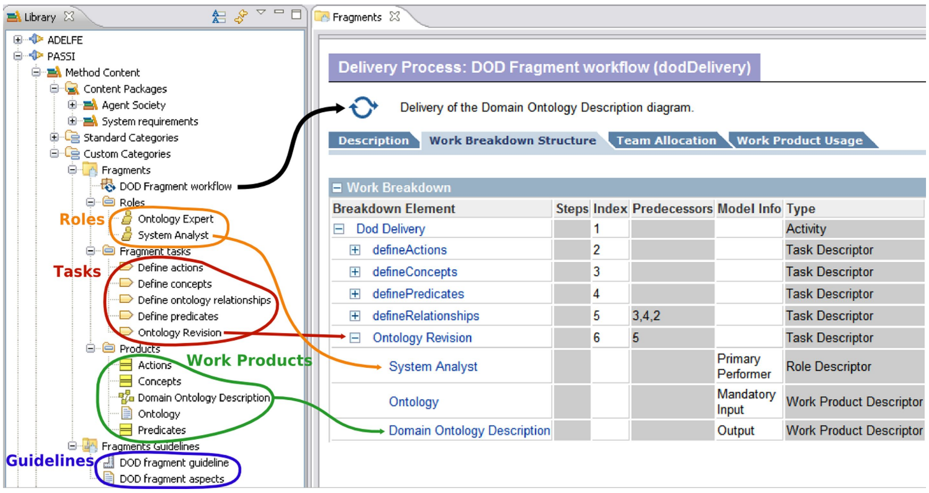
Custom Categories. Method plugin allows the re-usability of its whole contents definitions and process uses. In order to ease the reuse of fragments, we propose to integrate their definition into two *Method Plugin* customs categories, *Fragments* and *Fragments Guidelines.* Those two categories are intended to group, on one hand, the work to do and the way it can be done (*Tasks* and *Process Patterns*) and, on the other hand, guidances (concepts and all useful documents for the use of fragments). Those categories could be themselves categorized thanks to inner custom categories (see 6.2).

Going Further. *Method Library* can be used as general purpose SEP container. It could even contain sets of reusable elements (content and process) that do not belong to a specific SEP, but could be used as building bricks. More than an agent-oriented methodologies repository, SPEM 2.0 *Method plugin* and *Library* allow to reuse any other method plugin elements since they have been imported in the same library. The EPF eclipse plugin is a SPEM 2.0 implementation which provides all those capabilities, thus it can be used to define method plugins into library as well as tailor new SEP from those plugins (*Method Configuration*). Projects such as OpenUP[3] meet the requirements of the FIPA methodology TC by using the modularity and re-usability skills of SPEM through the EPF plugin. In fact, OpenUP provides an open-source, common and extensible base for iterative incremental SEP. It seems obvious that AOSE will make profit of such projects by defining specific AO method plugins and reusing predefined ones.

6 SPEM 2.0 Fragments Definition

This section presents a kind of fragment definition "process", that is illustrated with ADELFE, and PASSI fragment. First of all, method plugins have to be defined. This allows us to describe re-usable method and process contents (*Tasks, Work Products, Roles* and *Guidance*, etc.). Then, some customization is needed. New plugins are equipped with two *Custom Categories*: *Fragments* and *Fragments Guidelines* which will gather all elements that are needed for the

[3] `http://epf.eclipse.org/wikis/openup`



- ADELFE
- PASSI
  - Method Content
    - Content Packages
      - Agent Society
      - System requirements
    - Standard Categories
    - Custom Categories
      - Fragments
        - DOD Fragment workflow
        - Roles
          - Ontology Expert
          - System Analyst
        - Fragment tasks
          - Define actions
          - Define concepts
          - Define ontology relationships
          - Define predicates
          - Ontology Revision
        - Products
          - Actions
          - Concepts
          - Domain Ontology Description
          - Ontology
          - Predicates
      - Fragments Guidelines
        - DOD fragment guideline
        - DOD fragment aspects

Delivery Process: DOD Fragment workflow (dodDelivery)

Delivery of the Domain Ontology Description diagram.

Work Breakdown

Breakdown Element	Steps	Index	Predecessors	Model Info	Type
Dod Delivery		1			Activity
defineActions		2			Task Descriptor
defineConcepts		3			Task Descriptor
definePredicates		4			Task Descriptor
defineRelationships		5	3,4,2		Task Descriptor
Ontology Revision		6	5		Task Descriptor
System Analyst				Primary Performer	Role Descriptor
Ontology				Mandatory Input	Work Product Descriptor
Domain Ontology Description				Output	Work Product Descriptor

Fig. 2. DOD PASSI fragment definition in SPEM 2.0 using EPF

description of re-usable method fragments. At this stage, the definition of fragments only depends on their characteristics (see section 5.1). The following presents SPEM 2.0 fragment definitions. ADELFE characteristic parts (previously identified as fragments) as first instance, as well as a PASSI fragment. We use the EPF eclipse plugin to realize these definitions.

6.1 ADELFE Fragments

ADELFE has been defined as a sequence of *Work Definitions* containing *Activities, Work Products* and *Role*. As this description could not be straight forward translated to SPEM 2.0, due to concern separation mainly, we have to gather basic work definitions (*Tasks* in SPEM 2.0) into some kinds of categories. This is the purpose of *Discipline*: the *Tasks* intended to be executed during a particular *Phase* or *Iteration* are grouped in the same *Discipline*. In this example, we focus on the fragment called: “Verify AMAS adequacy” *Task*. As a SPEM 2.0 *Task*, it can be re-used and even extended in other method plugin. It consists of two *Steps*: “Verify adequacy at the global” and “local level”. This *Task*/fragment is related to specific *Guidances*, which ease its use and integration within an other method: the “AMAS Adequacy” *Concept* and the “Adequacy tool” *Tool Mentor*[4]. We have added this task at the *Fragments* category, as well as its related guidances to the *Fragments Guidelines* category (as shown in the right panel of the screen shot in figure 2).

[4] *Concept* and *Tool Mentor* are both *Kind* of *Guidance*, they are defined in the SPEM 2.0 base plugin.

6.2 PASSI DOD Fragment

We choose the “Domain Ontology Description Fragment” as an other instance of SPEM 2.0 fragment description. According to its definition [5], it is a “coarse grain” fragment. It involves several tasks, work products in a specific workflow. We have made the choice of mapping it to a *Process With Method* element: a *Delivery Process*[5] which contains an *Activity* named “DOD delivery”. This activity is broken down into tasks leading to the delivery of the DOD. We also have defined some inner categories to distinguish tasks from process, roles and products (see figure 2). Moreover, the DOD PASSI fragment defines “fragment aspects” that haven’t been defined in ADELFE fragment. We have determined that this concepts should mapped to a “generic kind of guidance”: *Supporting material* as it is used to contain “information not specifically covered by the other guidance types”.

7 Discussion and Prospects

According to the previous examples, it seems that SPEM 2.0, thanks to its capabilities (as shown in section 4.2), goes one step further toward the provision of an agent-oriented *Method library* where all the identified methodology fragment should be defined and stored as reusable and extensible *Method Content* and *Processes* within *Method plugins*. Moreover, the SPEM standard provides a common frame or language to define methods, good practices and gained expertise in the field of AOSE, as well as a widespread use and tool support for the description, management or even enactment of processes.

Situational Method Engineering has promoted for years, the idea that not a single method could fit all method engineers needs, it strongly depends on the studied domain. From this idea, FIPA has proposed the concept of fragments which is intended to cope with this problem. We assume that the SPEM 2.0 version has reached a stage where modularity and re-usability area is sufficient to allow fragments definition. However, even if a descriptive specification is already possible using SPEM, some important issues still have to be faced. For instance, we have presented the description of fragments with EPF, this is not straightforward. In fact, *Process Component* is not implemented in EPF, methodological units can’t be expressed so easily with encapsulation and interfaces (provided or requested). Although a task can be defined with input and output work products (as we discussed it in section 5.2) the SPEM 2.0 base plug-in provided by EPF does not allow to define coarse-grained encapsulated units.

Therefore, what about the assembly of such fragment? How can we fix problems such as input/output type verifications, name conflicts, etc.? How can we assist method engineer during the election of the more suitable fragment while they are devoted to different agent paradigms? One of the ideas proposed in [20] for the browsing and search into fragment repository, is the use of an ontology, or at least

[5] SPEM 2.0 base plugin special kind of *Process Package*.

a taxonomy, containing the concepts linked to each fragment and which can be use as a guide for the tailoring of a new fragment-based methodology.

Fragments interest is precisely that they focus on specific agent paradigms (ADELFE for emergent system and cooperative agent, ASPECS for holonic MAS, etc.). Thus, they depend on the underpinning method meta-model from which they are derived. This implies that, to perform fragment connection to another, a kind of mapping or “glue” have to be created because related concepts may belong to different paradigm or domain. Thus, building ontology over those concepts may ease this “glue” generation, maybe using model transformation. However, if we reconsider the ADELFE fragment of adequacy verification, is it really specific to AMAS or could it be generalized to any type of agent? Therefore, could it be used to verify adequacy of a system with holonic paradigm, provided that it has been extended or adapted. This implies that fragments should be parameterized with concepts like the type of agent or system (cooperative, BDI, holonic, etc.). For instance, the adequacy verification will become: “verification of the adequacy between solution and the problem domain”.

However, assembly is a well-known problem in the scope of programming language, composing software component or aspects is not trivial. Therefore, composing methodology fragments appears to be a complicated task, thus automate the design of a new methodology will result in a much more complicated work.

This paper has presented some preliminary steps towards this further goal. The main work will be to propose tools to combine these fragments. Because it is a complex problem requiring adaptation, we can propose an adaptive multi-agent system to solve it where the fragments will be agentified.

References

1. Bergenti, F., Gleizes, M.-P., Zambonelli, F. (eds.): Methodologies and Software Engineering for Agent Sytems. Kluwer Academic Publishers, Dordrecht (2004)
2. Bernon, C., Camps, V., Gleizes, M.-P., Picard, G.: Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 172–202. Idea Group Pub., USA (2005)
3. Capera, D., Georgé, J.-P., Gleizes, M.-P., Glize, P.: The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents. In: TAPOCS 2003 at WETICE 2003, Linz, Austria. IEEE CS, Los Alamitos (2003)
4. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems 27(6) (2002)
5. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardisation to research. Int. J. of Agent-Oriented Software Engineering 1, 91–121 (2007)
6. Cossentino, M., Gaglio, S., Sabatucci, L., Seidita, V.: The PASSI and agile PASSI MAS meta-models compared with a unifying proposal. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS, vol. 3690, pp. 183–192. Springer, Heidelberg (2005)
7. Cossentino, M., Gaud, N., Galland, S., Hilaire, V., Koukam, A.: A holonic metamodel for agent-oriented analysis and design. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS, vol. 4659, pp. 237–246. Springer, Heidelberg (2007)

8. FIPA. Method fragment definition, fipa document edition (November 2003)
9. Gomez-Sanz, J., Pavon, J.: Agent Oriented Software Engineering with INGENIAS. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS, vol. 2691, p. 394. Springer, Heidelberg (2003)
10. Henderson-Sellers, B.: Evaluating the feasibility of method engineering for the creation of agent-oriented methodologies. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS, vol. 3690, pp. 142–152. Springer, Heidelberg (2005)
11. Juan, T., Sterling, L., Martelli, M., Mascardi, V.: Customizing AOSE methodologies by reusing AOSE features. In: Rosenschein, J.S., Sandholm, T., Wooldridge, M., Yokoo, M. (eds.) International Conference on Autonomas Agents and Multi-Agent Systems (AAMAS 2003), Melbourne, Australia, pp. 1024–1025 (2003)
12. Luck, M., Ashri, R., d'Inverno, M.: Agent-Based Software Development. Artech House, Inc., Norwood (2004)
13. Object Management Group, Inc. Meta Object Facility (MOF) 2.0 Core Specification (October 2003)
14. Object Management Group, Inc. Unified Modeling Language (UML) 2.0 Infrastructure Specification. Final Adopted Specification (August 2003)
15. Object Management Group, Inc. Unified Modeling Language (UML) 2.0 Superstructure Specification. Final Adopted Specification (August 2003)
16. Object Management Group, Inc. Software & Systems Process Engineering Metamodel Specification v2.0, omg edition (October 2007)
17. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 185–193. Springer, Heidelberg (2001)
18. Padgham, L., Winikoff, M.: Prometheus: A methodology for developing intelligent agents. In: Proceedings of the Third International Workshop on Agent Oriented Software Engineering at AAMAS (July 2002)
19. Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M.-P.: Model Driven Engineering for Designing Adaptive Multi-Agent Systems. In: Artikis, A., O'Hare, G.M.P., Stathis, K., Vouros, G. (eds.) ESAW 2007. LNCS, vol. 4995, Springer, Heidelberg (2008)
20. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent system design. In: Paoli, F.D., Stefano, A.D., Omicini, A., Santoro, C. (eds.) Proceedings of the 7th WOA 2006 Workshop From Objects to Agents, September 2006. CEUR Workshop Proceedings, vol. 204, CEUR-WS.org (2006)
21. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming, 2nd edn. ACM Press/Addison-Wesley (2002)
22. Wooldridge, M., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems 3(3), 285–312 (2000); Times Cited: 3 Article English Cited References Count: 34 412fd

A MAS Metamodel-Driven Approach to Process Fragments Selection

M. Cossentino[1,2], S. Gaglio[1,3], S. Galland[2], N. Gaud[2], V. Hilaire[2], A. Koukam[2] and V. Seidita[3]

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, Palermo, Italy
cossentino@pa.icar.cnr.it

[2] Systems and Transport Laboratory (SeT) - Belfort, France
{stephane.galland,nicolas.gaud,vincent.hilaire,abder.koukam}@utbm.fr

[3] DINFO - Università degli studi di Palermo, Palermo, Italy
{gaglio,seidita}@dinfo.unipa.it

Abstract. The construction of ad-hoc design processes is more and more required today. In this paper we present our approach for the construction of a new design process following the Situational Method Engineering paradigm. We mainly focus on the selection and assembly activities on the base of what we consider a key element in agent design processes: the MAS metamodel. The paper presents an algorithm establishing a priority order in the realization (instantiation) of MAS metamodel elements by the fragments that will compose the new process.

1 Introduction

Multi-Agent systems metamodels (MMMs henceafter) and the composition of new design process achieved, in the last years, a greater attention in the agent community. As regards MMMs, the growing importance of Model Driven Engineering approaches required a great effort in the study and modelling of systems on the basis of their metamodels. Besides the effort spent on studying techniques, methods and tools for the production of the right design process meeting specific process requirements (ad-hoc design process for specific situation and development context for solving a specific class of problems), is today more and more increasing. In this field, Situational Method Engineering (SME) [1], provides means for constructing ad-hoc Software Engineering Processes (SEP) by following an approach based on the reuse of portions of existing design processes (often called method fragments[1]). Our work is mainly focused on the use of SME [2–4] for the construction of customized agent-oriented design processes.

In this paper, we show the importance of the MMM in the selection of fragments that will constitute the new SEP, and we explore how MMM could guide in the selection and assembly phases when a new design process is under construction. Selection of fragments is tightly related to the identification of the

[1] From now on in this paper we will use the term *Process Fragment* or simply *Fragment*

new process requirements; fragments included in this process should, in fact, concur to the satisfaction of such requirements. Our thesis is that (some of the) new process requirements are linked to the MAS metamodel elements (MMME) instantiated by the fragments. More details on the structure of a fragment in our approach can be found in [5].

In order to exemplify the fact that many requirements have a natural relationship with some MMMEs, it is sufficient to think about the desired adoption of a goal-oriented analysis rather than an interaction-based one (using use cases). Such different desires will directly bring to the presence of different elements in the MMM (goals rather than use cases). Several other examples can be reported to support this point but we agree that not all requirements can be related to one or more MMME, for instance requirements related to the adoption of a specific practice in performing early requirements analysis will always produce the same output (and therefore will be related to the same MMMEs) but in a different way. In this paper we will only briefly discuss the identification of the MMME starting from the list of the new process requirements. This argument is out of the scope of this paper that it rather focussed on the fragments selection phase and the role of the defined MMM in it. Besides, fragments assembly (that follows fragments selection) is naturally related to the structure of the MMM too, since according to the relationships defined in the MMM, fragments instantiating some elements will naturally need as an input some elements and will produce, as an output, some others that are required elsewhere.

Our proposal consists in the definition of a prioritization algorithm that is used to create an ordered list of MMMEs. This list will be used to select the fragments: the first element of this list will lead to the selection of the first fragment to be imported in the new process and so on for the others (this does not necessarily mean that the fragment related to the first MMME will be positioned at the beginning of the new process life-cycle).

This algorithm establishes an important guideline for method engineers; in other approaches, the selection of fragments is strongly related to method engineer experience [6] or the adoption of complex deontic matrices [7]. The selection of each fragment automatically generates constraints on the following ones, thus giving a great importance to the order used to select them. We are convinced this caused the diffused feeling that method engineering is such a complex discipline that its usefulness is quite limited. With our approach, method engineers do not need a great experience or the capacity to use complex matrices, they only need a well structured repository where MMME can be used for fragments retrieval (we presented such a structure in [8]) and the guidelines we propose in this paper (as already said some activities, namely MMM definition starting from requirements and fragments assembly, are only briefly discussed here, the focus will be on fragments selection).

This article also reports an experiment of creation of a new process (called ASPECS[2]); this is not a classical toy problem but rather we are dealing with the construction of a large process for the design of large multi-agent systems.

[2] ASPECS: Agent-oriented Software Process for Engineering Complex Systems.

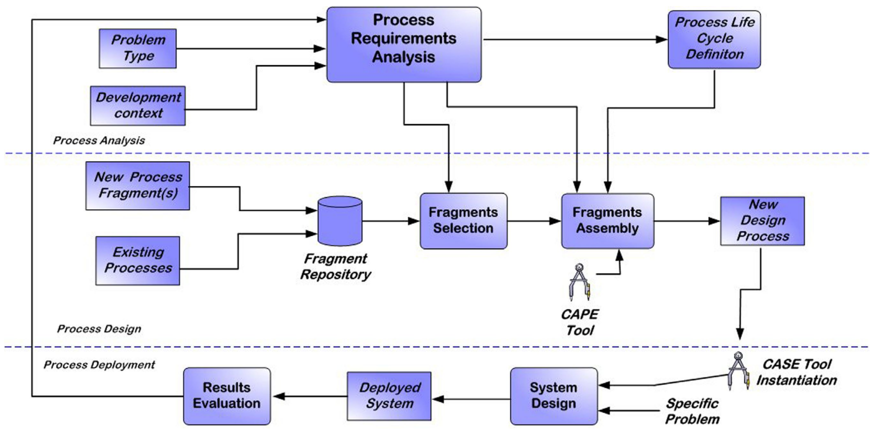


Fig. 1. The PRODE approach for design process composition

The MAS metamodel of this new process [9] is mainly composed by elements coming from the PASSI [10] and CRIO [11] existing design processes and supports Janus as an implementation platforms for holonic agents.

The paper is organized as follows: the next section gives a brief description of the proposed approach. Section 3 lists the requirements from which we started for developing the new process, the experiment done and quickly overview the resulting ASPECS design process. Section 4 discusses similar approaches in the field and, finally, some conclusion statements are provided in section 5.

2 The Proposed Approach

The contribution we propose in this paper is a portion of a complete approach to agent-oriented design process composition. In order to better position this contribution we now describe the overall approach named PRODE (PROcess DEsign for design processes).The PRODE approach is organized in three main phases (see Figure 1): process analysis, process design and process deployment.

Process Analysis deals with requirements elicitation and analysis of the process to be developed. It produces a set of elements, mainly a portion of the MMM, affecting the Process Fragments Selection and Assembly activities. Finally in the Process Deployment phase the new SEP is instantiated, used to solve a problem and then evaluated. Evaluation results are useful for defining new requirements for the next SEP (if any) or improving the designed one. It is worth to note that we consider the process of defining a new design process as an iterative and incremental one.

Process Requirements Analysis is the first activity a method designer undertakes in his work. It has inputs coming from the type of problem to solve. The new process has in fact to be tuned for: *(i)* a specific solution strategy to a class of problem, *(ii)* the development context (composed by the available resources such as people, tools, coding/modelling languages and platforms), and *(iii)* competencies that are available in the SEP enactment group. This activity generates the system metamodel and the other process elements (available stakeholders, required activities/work products) used for the new process creation.

The metamodel contains all the concepts and their relationships. It can be used to design and describe the system under study. It is organized in three different domains, each one being associated to a phase of the development process. The first domain is dedicated to the analysis and provides concepts to describe the problem independently of a given solution. The second provides concepts for the design of a solution independently of a given implementation. And the last one provides platform-specific concepts.

We assume that each concept of the metamodel will be defined (instantiated) in at least one fragment of the process whereas it can be related to other MMME or cited in several fragments. The list of MMMEs is used for the *process fragments selection* from the repository [8][5] as it will be discussed in the next sections. In the *Process Life Cycle Definition* activity, these inputs are also used to define the process life cycle that establishes the structure the designer has to follow during process fragments assembly.

In the **Process Design** phase, process fragments are extracted from existing design processes (or created from scratch) and stored in the Fragment Repository.

In the *Fragments Selection* activity the method engineer adopts the algorithm described in the next subsection for selecting fragments in the prescribed order. The *process fragments assembly* activity results in the new SEP. This activity consists in putting together the selected process fragments according to the structure of the previously identified process life cycle.

This activity is still one of the most important unsolved points in the SME field and some proposal have been done in [12][13]. It is a very complex work where the method designer has to collate all the elements gathered in the previous activities and to merge them by using his experience and skills.

During the *Process Deployment* phase, the system designer adopts the new design process with the help of a CASE tool for solving a specific problem (we developed the Metameth application for such a task). After that the designed system is used and experimented, an evaluation activity occurs in order to evaluate the new design process; gathered information can be used to identify new process requirements for a next iteration.

In section 3 an example on how we apply this process is provided by referring to the construction of the ASPECS process.

In this paper the main focus is on the use of a core part of the MAS metamodel as a guide towards the selection of fragments. The procedure we defined (Figure 2) starts from the identification of the core part of the MAS metamodel that is done by evaluating the contributions that could come from existing design processes or development platforms (in our case they were PASSI, CRIO, JANUS because of our past experiences with them). In fact it is logical to expect that people already skilled with the concepts related to some existing processes or platforms prefers

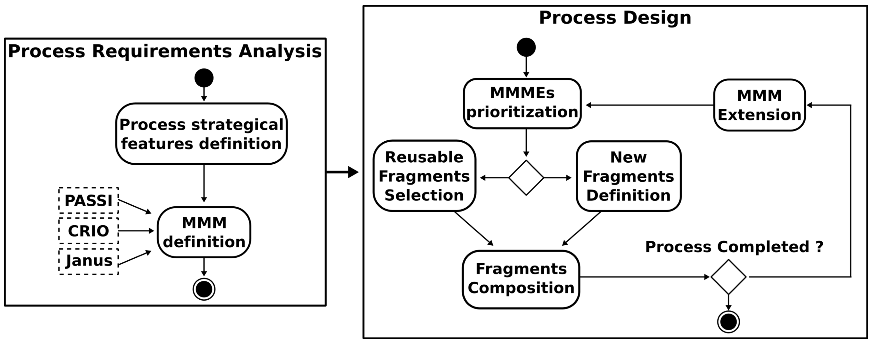


Fig. 2. Details of theprocess requirements analysis and process design phases presented in Figure 1

to reuse them rather than to build everything from scratch. Parts of those metamodels have been reused in order to satisfy the new process requirements that will be described in the experimental part of the paper (section 3).

In the following subsections we discuss the most important steps of this process: the new MAS metamodel construction and the new process design phase where fragments are retrieved from repository and assembled.

2.1 MAS Metamodel Elements Prioritization

As already said, in this work we composed the new metamodel on the basis of portions of metamodels coming from PASSI, CRIO and Janus. In so doing we are aware that defining the core MAS metamodel means defining a relevant part of the 'philosophy' that will be behind the new design process. For this reason we performed this activity during meetings involving stakeholders. We tried to deduct the list of elements by the portions of the cited processes that could satisfy the new process requirements. Of course this was not sufficient and it was therefore necessary to add new concepts for dealing with the specific case. For instance a lot of work has been done in the organizational definition of the agent society holonic structure as well as on the specification of possible roles that could be played by agents inside an holon (Head, Representative, Part/Multipart and StandAlone). These are crucial choices that conditioned the entire process and they have been largely debated before adoption. Some details about the definition of the core metamodel will be provided in the experimental part of the paper in section 3.

The work for designing the new process based on the defined core metamodel can be represented as a cycle composed of three sub-phases as illustrated in Figure 2: *(i)* prioritization of MAS Metamodel Elements (MMMEs); *(ii)* identification and assembly of process fragments defining the MMMEs; *(iii)* extension of the metamodel until the complete process is defined.

The process is detailed in the following algorithm:

```
//Sub-phase 1: MAS metamodel elements prioritization
1.  Select a metamodel domain and consider the resulting portion of metamodel as a graph
      with nodes (MMMEs) and links (relationships);
2.  Define List_elements as a list whose elements are lists of MMMEs and associated
      priority p: List_elements (p);
        a.  p<-1;
        b.  List_elements <- null;
3.  Produce a linearization of the MMMEs nodes according to a topological sort (elements
      with fewer relationships first) in List_elements, p is the priority index of each
      node in the list
// Sub-phase 2: Selection/Assembly of fragments related to the core MAS metamodel
4.  Select/Build fragments for defining (i.e. instantiating) the selected MMMEs by doing:
        a.  p<-1;
        b.  Selected_el<-List_elements.select(p);
        c.  While Selected_el.count>0 do{
            c1.    identify a reusable fragment for the instantiating the first element of
                 Selected_el or create a new one.
            c2. Selected_el.RemoveFirst}
        d.  Increment p.
e.  Repeat from b.
5.  Assembly the fragment in the new process (eventually modify it if required)
6.  Select the next metamodel domain (if any) and repeat from 2
//Subphase 3: MAS Metamodel Extension
7.  If the process is not completed (i.e. not all design activities from requirements
      elicitation to coding, testing and deployment have been defined)
        a.  Introduce new MMMEs
        b.  Repeat from 1.
```

The algorithm prescribes (point 3) a linearization of the list of elements of the MMM according to a topological sort criterion. The guideline we propose for accomplishing this task consists in a few steps: consider the elements of the graph, initialize priority p=1, select the element(s) that has fewer relationships with the others, remove it/them (and insert it/them in the List_elements list with priority p), remove the element(s) and all the related relationships from the metamodel, increment p, iterate.

Figure 3 reports an example of application of the proposed algorithm; here a portion of the ASPECS core metamodel is reported, the agency domain. Applying the proposed algorithm we can see that, at the first step, the elements with less relationships are: *Capacity* and *Message*; both these are assigned a level of priority p equal to 1. The next step is to remove these elements from the metamodel (see the right part of Figure 3) and iterate the procedure. The next elements to be considered (p=2) are *Communication* and *Service*, they both have one relationship with *AgentRole*; we can insert them in the List_elements list and remove from the metamodel. Further steps are omitted because of space concerns.

The extension of the core MAS metamodel towards the completion of the process obtained by composing fragments and it should be strongly affected by the awareness of the new process requirements and the relationships among requirements and MMMEs. In extending the initial core metamodel some other criteria should be considered as well. First, the opportunity of reusing existing fragments could lead to the introduction of specific MMMEs related to them.

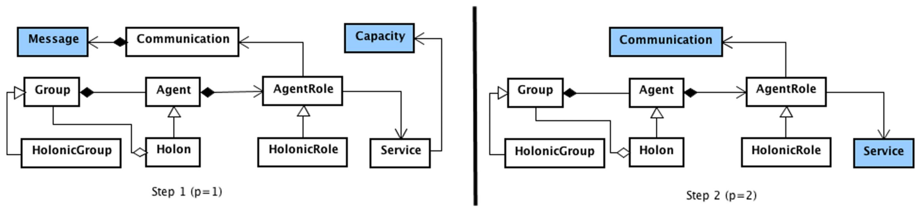


Fig. 3. The first two steps in the prioritization of the Agency Domain elements

This is a kind of bottom-up criterion that privileges the reuse of well-known and tested fragments. Second, as a consequence of adopting a Model Driven Engineering (MDE) approach in the development of ASPECS, we think that: (i) the three identified MAS metamodel domains may be regarded as the three different MDE models; (ii) elements belonging to a domain should have a correspondent element in the following one (correspondence is realized by the transformation from one model to the other). The second rule can have some exceptions related to specific cases when an element is regarded as a design abstraction useful at one specific level but it is not forwarded to the next one. A detailed discussion of criteria and guidelines for MAS metamodel extension is out of the scope of the paper and will be omitted.

3 Building ASPECS

In this section we describe the process we adopted for building ASPECS. We report the process requirements, the initially created core metamodel, the definition of the precedence order of the metamodel elements, the selection/assembly of process fragments and the extension of the metamodel with the consequent selection of new fragments in an iterative process. Finally a short description of the resulting process is provided.

3.1 Requirements for the Construction of ASPECS

The design of the ASPECS methodology has been constrained by a set of requirements that according to the inputs of the process requirements analysis phase presented in Figure 1, can be classified as follows:

(i) *Problem Type*: the scope of the new design process was to develop very large MASs for the solution of complex problems suitable for an hierarchical decomposition.
(ii) *Development context*: the development of the ASPECS methodology can be seen as a joint work of people coming from two different experiences: people working at the SET laboratory who had a strong background in the design and implementation of holonic systems with a strong accent on organizational aspects of MASs (CRIO process) and one new lab member who was the main author of

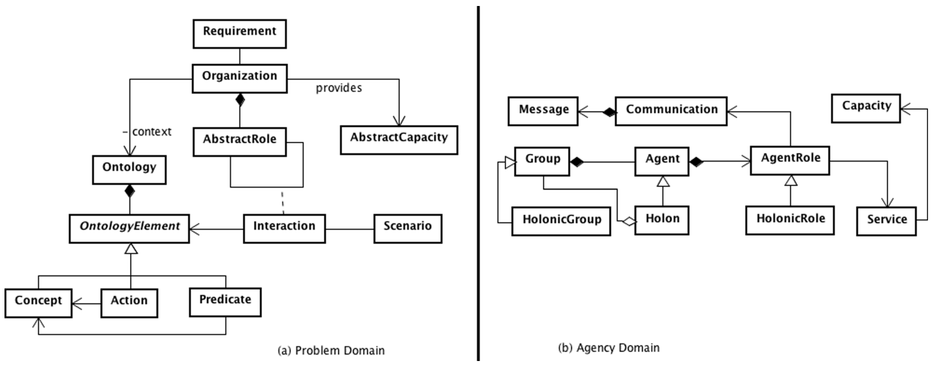


Fig. 4. A part of the ASPECS Problem and Agency domains core metamodel

a process (PASSI, [10]) for the design of MASs where agents were mostly peers and whose important features were: the use of ontologies, a requirements-driven agent identification, the adoption of patterns and tools for supporting design/-coding activities. Participants to this project soon agreed to preserve some key elements of their backgrounds and skills in order to enable an easier transition to the new design process. As regards agents implementation, in the SET lab, the development of a new coding platform Janus was undergoing and its adoption in the new design process was, of course, highly desirable.

These requirements concurred to the definition of the process we describe in the next subsection.

3.2 The Core Metamodel

A part of the initial core metamodel defined for the ASPECS process can be seen in Figures 4(a) and 4(b). This metamodel is a consequence of the process requirements and design choices done during several meetings. Just in order to exemplify how the metamodel was defined we can consider two of the process requirements:

1. An organizational approach was desired were a direct link could be established between the problem and the organization that solves it. This was expected to generate solutions where the (hierarchical) organization structure was an evident decomposition of the problem.
2. A FIPA-compliant communications structure was required.

The first requirement, in the design team opinion, finds a solution in the adoption of a direct link between the Organization and Requirement MMMEs (see Figure 4(a)). This link would represent the direct correspondence between the requirements and the organization that would fulfil them.

The remaining part of the reported Problem Domain core metamodel descends from the definition of organization we decided to adopt: *An organization is defined by a collection of roles that take part in systematic institutionalised patterns of interactions with other roles in a common context. This context consists in shared knowledge and social rules/norms, social feelings, etc and is defined according to an ontology. The aim of an organisation is to fulfil some requirements.*

This definition largely comes from the CRIO approach but it has also been enriched with concepts coming from PASSI (for instance we decided to describe the organizations context by using an ontology). This is coherent with the general requisite of reusing teams members experience as much as possible. The last sentence of the definition is the consequence of the above reported new process requisites.

The second requisite naturally brings to the portion of metamodel reported in Figure 4(b). This was largely inspired by a corresponding portion of the PASSI metamodel where roles, communications, messages are connected in order to satisfy FIPA specifications.

From these and other similar considerations we built the core metamodel for the ASPECS process.

Summarizing, the core metamodel definition process is mainly composed by the following steps: analysis of the process requirements, identification of MMMEs and/or their relationships that could concur to these requirements satisfaction, modification of the core metamodel according to strategic design choice (for instance the adopted definition of Organization reported above).

In the next subsection we discuss the prioritization of the MMMEs representing the order we expect to instantiate these elements in the fragments that will compose the new design process.

3.3 Prioritization of MAS Metamodel Elements

The priority order of the MMMEs was defined by applying the already discussed heuristics to the Problem domain metamodel reported in Figure 4(a). The resulting list is: Requirement, AbstractCapacity, Scenario, AbstractRole, Interaction, Organization, Action, Predicate, Concept, Ontology.

Similarly we obtained a priority order list for the MMMEs elements of the following domains (Agency and Solution). The MAS metamodel of the Agency domain is reported in 4(b).

After this step it is possible to select of fragments from the repository or the construction of new ones in order to define the elements according to the prescribed order. This process will be discussed in the next subsection.

3.4 Selection of Fragments

In performing the fragments selection activity, we refer to our repository of fragments [8]; it includes fragments extracted from PASSI, Agile PASSI, TROPOS, and Adelfe. For the presented experiment we used only fragments coming form PASSI and we purposefully prepared the documentation of fragments coming from CRIO that were not in the repository. Since several MMMEs required by this novel approach (for instance Holon) are not defined by fragments in the repository, we expect to produce several new process fragments, hoping of reusing and modifying some existing ones when possible.

According to the previous discussed list of MMMEs, the first retrieved process fragment is supposed to instantiate the Requirement MMME, a model of system requirements by starting from text usage scenarios. This is exactly what the Domain Requirements Description fragment of PASSI does and it was thus reused. The second MMME (AbstractCapacity) is instantiated by the Capacity Identification fragment reused from CRIO. It is interesting to note that there is no real difference in the precedence order of these two first elements (they share the same value of priority). It is therefore not important to start from one or the other. As already discussed, the two fragments we identified are not necessarily the first two of the process life-cycle. This order in facts arises from the mutual dependencies in terms of input/output among all fragments and could be determined, for instance, by drawing a dependency diagram.

The realization of the third MMME (Scenario) presents an interesting issue: this element is defined in the PASSI Role Identification fragment (where some sequence diagrams are used to describe agent interactions within scenarios). This fragment operates on several different MMMEs: Agent, Role, Actor, Message and its output is the required Scenario. As it can be seen, some of these elements are not part of the core metamodel. This situation (that is quite common) can be solved in two ways: *(i)* the fragment is modified, *(ii)* the elements are added to the metamodel thus enlarging the structure defined in the initial core. Further details on the extension of the core metamodels will be presented in the next subsection.

In a similar way we defined the remaining part of the process. In this discussion we omitted the details of each fragment and the difficulties found in defining the new ones as well as in modifying the reused ones while adapting them to cope with the new specific issues.

In the next subsection we discuss some examples of extension of the initial core MAS meta-model done in order to refine the initial sketch of the process.

3.5 Completion of the Process and Extension of the Core Metamodel

We view the construction of a new design process as an iterative-incremental activity that can be decomposed in the following steps: (i) Construction of a process stub including several fragments. (ii) Test of the process portion. (iii) Evaluation of results. (iv) Next iteration planning in terms of new process requirements to be addressed, changes to be done in the existing process stub, and new parts of the metamodel to be included in the process.

In the ASPECS design process, we performed the first significant test after completing a draft of the System Requirements phase. This test consisted in using the new design process stub for designing a couple of simple applications. As a result of this evaluation, we proposed one change: the explicit introduction of non-functional requirements in the early stages of the process (this implied an extension of the metamodel). After that, according to the 4-steps process discussed at the beginning of this subsection, we designed a new portion of the metamodel, more specifically, the core part of the Agency domain metamodel (see Figure 4(b)). We are not going to detail the work done on this part of the process, we will only discuss one interesting point: the extension of the initially defined core metamodel represented in Figure 4(b) to cope with some new process requirements identified during the iteration. After some evaluations, we realized that in the new process it was not possible to represent not FIPA-compliant agent interactions (for instance environment mediated). They had not been initially listed among the new process requirements but they were already supported by the Janus platform and sometimes used in previous projects developed in the lab. Another issue arose from the consideration that it was not possible to design simple (non holonic) agents like the conventional PASSI ones. This limited the possibility of integrating in the same design, complex holonic hierarchies with simple agents (devoted to deal with minor parts of the problem). In order to solve these issues we changed and extended the core metamodel by including a Conversation and an AtomicAgent MMMEs.

The extended metamodel has been fully realized by a set of fragments and then the process stub tested and evaluated as already described. The work continued in an iterative way until the complete process was defined and thoroughly tested.

Next subsection provides a short description of the resulting ASPECS process[3].

3.6 The Resulting Design Process

ASPECS combines an organizational approach with an holonic perspective. Its target scope can be found in complex systems and especially hierarchical complex systems. The principle of ASPECS consists in analyzing and decomposing the structure of complex systems by means of an hierarchical decomposition. The ASPECS process consists in four phases that are briefly described below.

The *Analysis* phase is based on the identification of a hierarchy of organizations whose global behaviour may represent the system under the chosen perspective. This phase starts with a requirements analysis activity where requirements are identified by using classical techniques such as use cases. Domain knowledge and vocabulary associated to the target application are then collected and explicitly described in the problem ontology. Each requirement is then associated to an organization that represents a global behaviour able to fulfil the associated requirements. The context of each organization is defined by a set of concepts of the problem ontology. The organization identification defines a first hierarchy of organizations that will then be extended and updated during the iterative process. The identified organizations are decomposed into a set of interacting sub-behaviours modelled by roles. The goal of a role is to contribute to the fulfilment of (a part of) the requirements of the organization within which it

[3] A complete description of the ASPECS process can be found at: `http://set.utbm.fr/index.php?pge=352&lang=fr`

is defined. In order to design modular and reusable organization models, roles should be specified without making any assumptions on the architecture of the agent that may play them. To meet this objective, the concept of capacity was introduced. A capacity is an abstract description of a know-how, a competence of an agent or a group of agents. The role requires certain skills to define its behaviour, which are modelled by capacity. The capacity can then be invoked in one of the tasks that comprise the behaviour of the role. In return, an entity that wants to access a role, should provide a concrete realization for each capacity the role requires.

The analysis phase ends with the capacity identification activity that aims at determining if a role requires a capacity. At this step a new iteration may possibly start. If all capacities required by roles at the lowest level of the hierarchy are considered to be manageable by atomic easy-to-implement entities, the process may stop.

The *Agent Society Design* phase aims at designing a society of agents whose global behaviour is able to provide an effective solution to the problem described in the previous phase and to satisfy associated requirements. The objective is, now, to provide a model of the agent society involved in the solution in terms of social interactions and dependencies among entities (holons and/or agents). Previously identified elements such as ontology, roles and interactions, are refined. At the end of the design phase, the hierarchical organization structure is mapped to a holarchy (hierarchy of holons) in charge of its execution. Each of the previously identified organizations is instantiated in terms of groups. Corresponding roles are then associated to holons or agents.

This last activity also aims at describing the various rules that govern the decision-making process enacted inside composed holons as well as the holons' dynamics in the system. All of these elements are finally merged to obtain the complete set of holons (composed or not) involved in the solution. In this way, the complete holarchy of the solution is described.

The *Implementation* phase aims at implementing the agent-oriented solution designed in the previous phase by adapting it to the chosen implementation platform, in our case, Janus. Based on Janus, the implementation phase details activities that allow the description of the solution architecture and the production of associated source code and tests. It also deals with the reuse of previously developed solutions.

The *Deployment* phase is the final one and it aims at detailing how to deploy an application over various Janus kernels. This phase starts with the description of the deployment configuration and details how the previously developed application will be concretely deployed; this includes studying distribution aspects, holons physical location(s) and their relationships with external devices and resources; organization and agent test activities complete the process.

4 Related Works

The work presented in this paper extended the Situational Method Engineering (SME) paradigm until now applied only to the Information System research field. In this section some conventional approaches will be discussed in order to provide an overview of related works.

There are different SME approaches in literature, all of them start by facing the three main phases for the construction of a new design process (process requirement analysis, fragments selection, fragments assembly) and all of them consider the fragment and the repository as fundamental elements. Despite of these works, the problem of how to select and to assemble a set of fragments is still an open issue.

An interesting approach is reported in [14]. Here the method designer firstly identifies the need for the new design process by carrying out a set of analysis activities on the application context. Basing on this analysis the method designer can identify and select a set of fragments that best fit the elicited needs and then he assemblies them. In this process the method designer uses a meta-modelling technique in order to model the design process by using class and data kinds of diagram.

The use of the meta-modelling technique is very similar to what we proposed in this paper, in fact it allows to model all the stages, activities and tasks that have to be performed in the new process thus creating a starting point for the method designer to identify the proper fragments. The main difference with the PRODE approach is that we use the metamodel in order to represent and define a complete knowledge for the new design process, and that its elements are the starting point for the selection from a repository and the assembly of fragments. This technique can be automated and, anyway, it reduces the relevance of designer skills in the process; this kind of dependence is one of the most important problems affecting every approach found in literature.

Another well known approach is proposed by Ralyté et al. [15][3]; the result of the previously said phases is a set of *maps*. These maps can be considered as a guideline during the development of the new design process; they are composed of three elements: source, intention and strategy. The method designer starts from the process requirements specification, through the maps he models the new process at different levels of abstraction and represents the method chunk, in so doing he is able to identify a set of method chunks that satisfy each requirement. Each chunk is equipped with a *descriptor*, done by a set of attributes (ID, name, type, application, etc.); this element lets the designer understand which method chunk can be used in each specific situation.

This kind of approach is quite different from the one we propose because it is more dependent on the method designer skills and knowledge but also it does not provide a knowledge description on the new design process and finally it is necessary to build and use a well defined chunk repository, as descends from the chunk definition and its specific form (the triplet and the descriptor).

Another approach, OPF (OPEN Process Framework) is based on the use of the so called Deontic Matrices [4][16]; after having identified the number and the type of activities to be performed, the workflow among them, the pre- and post-conditions and a first draft of lifecycle, the method designer uses a set of deontic matrices in order to find the relationships among fragments in pair and to be able to select a useful set of fragments. The most important difference and the most important value of this approach is that a very huge repository exists for that and this fact together with the use of deontic matrices allows the designer to cover a well defined path from the first selected fragment to the latest one through the established lifecycle giving, in this way, a real aid to the designer's work. The major problem is the deep knowledge required about the repository that, containing a large amount of fragments, surely cannot be easily acquired nor shared.

The approach presented in this paper has this kind of problem too, but the repository it is related to contains a reduced number of fragments, at an higher level of granularity and with the aid of the illustrated algorithm we claim that it is easier to select the right fragments for assembly.

Finally all the approaches, until now presented, greatly depend on the used definition of fragment our work, instead, aims at providing general methods and techniques that are customizable for every kind of application.

5 Conclusion

Based on the Situational Method Engineering, this paper reports an experiment of creation of a new process called ASPECS. The proposed approach starts from the identification of the new process requirements in terms of development context and problem type. The requirements are used for defining an initial core version of the MAS metamodel. The elements of this metamodel are then ordered in a precedence list, and in this order the fragments are retrieved from the repository and assembled in the new process. The resulting MAS metamodel of ASPECS [9] is mainly composed by elements coming from the PASSI [10] and CRIO [11] existing design processes and supports Janus as an implementation platforms of holonic agents. In previous works applying SME, the method engineer usually selects a set of process fragments that he considers as the best for fitting a particular situation and then modifies or adapts them. The approach described in this paper is different and it aims at being as much free as possible from the designer skills by providing a set of reusable guidelines for fragments selection and assembly.

References

1. Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young Management Consultants (1997)
2. Brinkkemper, S., Welke, R., Lyytinen, K.: Method Engineering: Principles of Method Construction and Tool Support. Springer, Heidelberg (1996)
3. Ralyté, J.: Towards situational methods for information systems development: engineering reusable method chunks. In: Procs.13th Int. Conf. on Information Systems Development. Advances in Theory, Practice and Education, pp. 271–282 (2004)
4. Henderson-Sellers, B.: Method engineering: Theory and practice. In: ISTA, pp. 13–23 (2006)

5. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardisation to research. International Journal of Agent-Oriented Software Engineering (IJAOSE) 1(1), 91–121 (2007)
6. Brinkkemper, S., Lyytinen, K., Welke, R.: Method engineering: Principles of method construction and tool support. International Federational for Information Processing 65, 336 (1996)
7. Firesmith, D., Henderson-Sellers, B.: The OPEN Process Framework: An Introduction. Addison-Wesley, Reading (2002)
8. Seidita, V., Cossentino, M., Gaglio, S.: A repository of fragments for agent systems design. In: Proc. Of the Workshop on Objects and Agents, WOA 2006 (2006)
9. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., Koukam, A.: A holonic metamodel for agent-oriented analysis and design. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS (LNAI), vol. 4659, pp. 237–246. Springer, Heidelberg (2007)
10. Cossentino, M.: From requirements to code with the PASSI methodology. In: Agent Oriented Methodologies, pp. 79–106. Idea Group Publishing, USA (2005)
11. Hilaire, V., Koukam, A., Gruer, P., Müller, J.: Formal specification and prototyping of multi-agent systems. In: Omicini, A., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2000. LNCS (LNAI), vol. 1972, p. 114. Springer, Heidelberg (2000)
12. Mirbel, I., Ralyté, J.: Situational method engineering: combining assembly-based and roadmap-driven approaches. Requirements Engineering 11(1), 58–78 (2006)
13. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-modelling based assembly techniques for situational method engineering. Information Systems 24 (1999)
14. van de Weerd, I., Brinkkemper, S., Souer, J., Versendaal, J.: A situational implementation method for web-based content management system-applications: Method engineering and validation in practice. In: Software Process: Improvement and Practice. John Wiley & Sons, Ltd., Chichester (2006)
15. Ralyté, J., Rolland, C.: An Approach for Method Reengineering. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, p. 471. Springer, Heidelberg (2001)
16. Henderson-Sellers, B.: Process Metamodelling and Process Construction: Examples Using the OPEN Process Framework (OPF). Annals of Software Engineering 14(1), 341–362 (2002)

An Evaluation Framework for MAS Modeling Languages Based on Metamodel Metrics

Iván García-Magariño, Jorge J. Gómez-Sanz, and Rubén Fuentes-Fernández

Dept. Software Engineering and Artificial Intelligence
Facultad de Informática
Universidad Complutense de Madrid, Spain
ivan_gmg@fdi.ucm.es, jjgomez@sip.ucm.es, ruben@fdi.ucm.es

Abstract. The fast pace of evolution in Agent-oriented Software Engineering leads to a great variety of continuously changing Multi-Agent System (MAS) Modeling Languages (MLs). In this situation, there is a rising need of evaluation for MAS MLs, as the plenty of works on this subject reflects. This paper follows this line of research presenting an evaluation framework to measure quantitatively MAS MLs. The framework includes metrics about availability, specificity, and expressiveness of the MLs. Otherwise than existing frameworks, this work considers metamodels to define its measures and focuses on the quantitative measurement instead of qualitative evaluations. With these metrics and the data gathered from existing MLs, the goal is to quantify the appropriateness of a given MAS ML for a particular problem domain. In addition, these metrics can quantitatively track the improvements of MAS MLs on these features. The paper also presents the results of the current experiments with the framework that have taken measures in nine problem domains with six MAS MLs.

Keywords: Metric, metamodel, modeling language, multi-agent system, evaluation.

1 Introduction

The agent field shares common conceptual foundations and principles but there is still a lot of discussion about how to crystallize them in concrete and agreed methodologies and modeling languages (MLs). For this reason, there is currently a large number of Multi-Agent System (MAS) methodologies, each one with its own modeling ML, like Tropos [3], INGENIAS [21], MaSE [7], Prometheus [20], PASSI [4], and Agile PASSI. There are also agent-oriented MLs which are not associated with any particular agent-oriented methodology, like AUML [2] or UML-AT [9]. Moreover, the aforementioned MLs are not frozen, but on the contrary, many of them keep evolving through the introduction of new concepts and improving their definitions. As a result of this situation, there is an increasing demand for the evaluation of MAS methodologies and MLs. This need crystallizes in the appearance of comparison and evaluation frameworks for agent-oriented MLs.

The existing evaluation frameworks for MAS MLs (see for instance [5, 23, 25, 26, 28]) have in common certain features. They focus in some aspects of the definition of the MLs. A ML is characterized by three components: the *abstract syntax* which is defined with a metamodel; the *semantics*; and the *concrete syntax* or notation. Not all the MLs define all these components, since its quite common not to provide the notation or even the semantics. This fact explains that existing frameworks focus mainly on the abstract syntax, less in semantics, and barely in the concrete syntax. Another common feature is that these frameworks usually adopt a qualitative approach to evaluation. Those with quantitative measures use discrete scales of few values (usually less than ten). Although these metrics are useful in their settings, they are not well-suited for some problems. For instance, when tracking improvements in the evolution of a ML as a consequence of the addition of new concepts, fine-grained metrics are necessary to quantify the effect of these additions on the overall language. The issue of how to compare the suitability for a problem domain of different MLs must also be considered. Most of frameworks rely on the subjective qualitative evaluation of experts for this problem. When looking for more precise measures, works in the metamodeling area provide clues on potentially relevant metrics. For instance, Kargl [13] defines a metric for the *explicitness* of modeling concepts in metamodels; Vépa [29] presents several metrics on KM3 metamodels about size, packaging, or inheritance reusability. Nevertheless, these metrics allow comparison between MLs but it is not clear how to use them to assess domain suitability.

The framework for the evaluation of MAS MLs in this paper addresses the previous issues with a set of quantitative metrics that consider the abstract syntax and the semantics of these MLs. These metrics are defined in terms of objective measures over the metamodels and ratings of experts about their associated semantics, specifically about whether an element is *necessary* or not. Given the metamodel of a ML and a domain problem, an element from the metamodel is said to be *necessary* in the domain if it is required for modeling in that domain and cannot be substituted with other elements of the metamodel.

The concept of *necessary* element is key in the framework, as it is the foundation of its tree metrics. The *availability* metric measures if the ML has all the elements necessary to model in a domain. The *specificity* considers if there are non-necessary elements in the ML. Finally, the *expressiveness* metric measures how many necessary elements are required to model a given problem. These metrics can assist the MAS designer in different evaluations. The availability and specificity metrics, which focus on the comparison between MLs, guide the choice of the appropriate MAS ML for a domain. The expressiveness metric allows tracking the progress in the expressive power of a ML along its evolution.

The structure of the remainder of this paper is as follows. Section 2 describes the evaluation framework with its three metamodel metrics. Section 3 and Section 4 report the measures gotten from several MAS MLs with the framework: Section 3 considers the *availability* and *specificity* metrics to compare MLs; Section 4 analyzes the improvements between versions of a MAS ML (the INGENIAS ML [21]) with the *expressiveness* metric. Section 5 reviews the related work. Finally, Section 6 discusses some conclusions and the future work.

2 Evaluation Framework

As stated in the introduction, this evaluation framework uses measurements on metamodels about the abstract syntax of the MLs and a limited evaluation of experts on certain semantic issues. This mixed approach tries to decrease the dependence of metrics on the experts' subjective evaluation. At the same time, the use of metamodels brings the advantages of an extensive software support, so it is possible to develop programs for measuring instead of manually inspecting the metamodels.

The framework currently includes three metrics, which are *availability, specificity*, and *expressiveness*. All of them are based on the semantic notion of the *necessary* concepts for a given problem domain, that is, those elements of the metamodel (i.e. entities, relationships, or others) required to express the concepts that appear in the domain. The availability and specificity metrics measure the suitability of the concepts of a ML for a particular problem domain. In contrast, the expressiveness metric measures the amount of instantiated elements required to represent the models of the problem domain. The following sub-sections describe these metrics.

2.1 Availability Metric

The goal of the *availability* metric is to measure the suitability of a ML for a particular problem domain. Given a particular domain, there are some concepts necessary to model its problems. The availability metric measures the percentage of these necessary concepts that are contained in the ML. The higher is the value obtained in the availability metric, the better is the conceptual framework of the ML for the domain. On the contrary, a low value indicates that the agent ML does not have concepts that developers considers necessary.

According to this introduction, the availability metric is defined as the ratio appearing in Equation 1. In this equation, *nc* indicates the number of necessary concepts for the modeling in a particular problem domain, and *ncmm* the number of these necessary concepts that are actually contained in the metamodel.

$$availability = \frac{ncmm}{nc} \tag{1}$$

The set of necessary concepts for a particular problem domain must be calculated according to the ML. The user must restrict to what the metamodel contains and decide which of the metamodel elements are necessary for solving the particular problem. Once this set is selected, the user must detect if there are concepts required to solve the problem that do not appear in the metamodel. These are the missing concepts. Thus, the set of necessary components is the union of the necessary components present in the metamodel and those missed. This observation allows an alternative formulation of Equation 1 as Equation 2, where (mc) is the number of missing elements and and $(ncmm)$ the number of necessary metamodel elements as before.

$$availability = \frac{ncmm}{ncmm + mc} \tag{2}$$

Given these equations, the best result for the availability metric is the unity (i.e. 100%). This result is obtained when there are no missing concepts $(mc = 0)$ in the ML.

2.2 Specificity Metric

The *specificity* metric measures the percentage of the modeling concepts in a ML that are actually used for a particular problem domain. If the value of this metric is low, it means that there are many non-necessary metamodel elements. Hence, the scope of the ML is probably more general than required for the domain. On the contrary, if the value of the specificity metric is high, the scope of the ML is targeted to that specific problem domain.

The specificity metric is defined with the Equation 3. This equation introduces a new term *cmm* that represents the number of concepts in the metamodel, both necessary and not. The specificity metric is the ratio of the metamodel concepts necessary for a problem domain and the whole number of metamodel elements.

$$specificity = \frac{ncmm}{cmm} \tag{3}$$

In the specificity metric, the best result is obtained when all the concepts of the metamodel are necessary $(ncmm = cmm)$ for the corresponding problem domain. In this case, the specificity value is the unity (i.e. 100%).

2.3 Maximizing the Availability and Specificity

A model designer should try to get high values in both availability and specificity metrics. Nevertheless, a high availability is commonly preferable to a high specificity. A high specificity implies that the language is very domain specific and the designer uses all its concepts, although this does not preclude that some required concepts may be missed. On the contrary, a high availability implies that the designer has all elements needed to model the problem, although there are also some not necessary. Theoretically, availability and specificity would get the highest scores when $ncmm = nc$ and $mc = 0$. Thus, a designer trying to optimize a ML with the maximum availability and specificity must select a subset of the ML with only those elements needed in the concrete problem.

The optimization of specificity and availability metrics has a relevant impact in the manipulation of the MLs. Consider the setting in which a company wants to develop a code generation engine that transforms the models expressed with a given ML into code suited for a specific platform. In order to guarantee the satisfaction of the requirements and minimize the cost of this project, the development team should select a subset of the metamodel of the mentioned ML with a high availability and specificity values. The high availability guarantees the effectiveness on tracking the modeling coverage of the new interpreter. The high specificity implies that the new code generator is implemented for as few elements as possible, that is, only for the smallest necessary subset of the ML.

A remark must be made here concerning so called *multi-perspective modeling.* A *multi-perspective* ML has several perspectives that represent pre-configured aspects of a system with a different focus and conceptualization. These perspectives usually overlap and the same features of a modeled system appear in different perspectives. This overlapping ease understanding of models but it also makes that these MLs [14] use more concepts than other MLs. For this reason, the multi-perspective MLs commonly get low results on specificity. However, this is not inherent to multi-perspective MLs as they can get high scores as long as all the perspectives are necessary. This is the case, for instance, of PASSI, which is a multi-perspective MAS ML [4] with high results for the specificity metric (see Section 3 for a comparison of MAS MLs in a group of problem domains).

2.4 Expressiveness Metric

The *expressiveness* metric measures the amount of model elements necessary to model a *system* in a problem domain. The less model elements that are necessary to specify the model, the more expressive is the ML. The term *system* denotes in this context any MAS that satisfy some specific requirements.

It is important to notice that the expressiveness metric is related to the metamodel instantiation process. The metric considers model elements instead of metamodel elements as the previous metrics. The model elements are those used in the specification of the model of a system. On the contrary, the metamodel elements represent concepts of the ML. Thus, the model elements are instances of the metamodel elements.

The expressiveness of a ML *ml* for a system *system* is defined with the Equation 4. The *nme* term denotes the number of model elements necessary for modeling the system of the problem domain. There are different metrics that can be considered for the size of the system, such as *lines of code* and *function points.*

$$expressiveness(ml, system) = \frac{size(system)}{nme} \tag{4}$$

The value of the expressiveness for a ML changes with the chosen metric for the system, what limits the value of this metric as an individual measure. However, this metric is relevant for the comparisons between MLs, since these relative values do not regard on the system size. Equation 5 shows the comparison of two MLs, denoted as *ml1* and *ml2*, with these metrics. In this equation, the *nme1* and *nme2* terms respectively denote the number of model elements for modeling the system with *ml1* and *ml2*. The most expressive ML is the one which needs fewer elements to represent the model of the system that solves the problem.

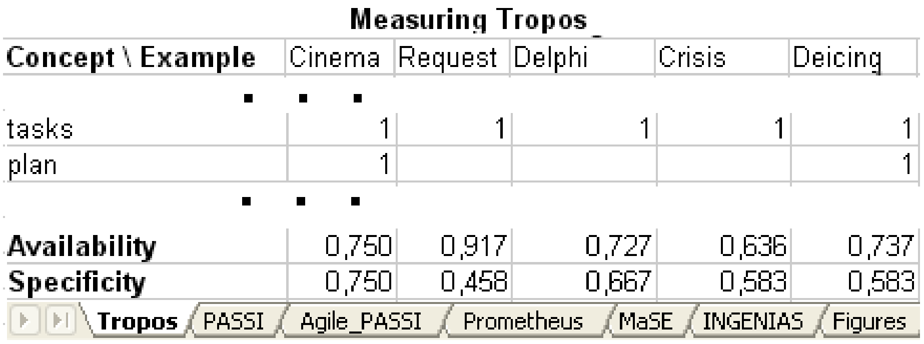
$$\frac{expressiveness(ml1, system)}{expressiveness(ml2, system)} = \frac{nme2}{nme1} \tag{5}$$

Besides comparing different MLs, the expressiveness metric is also useful to evaluate the improvements in the expressiveness of a given ML. If a ML undertakes some changes in its metamodel, its new expressiveness value can be compared with the old one. If the ratio is greater than one, the expressiveness of the ML has improved. For instance, a ML that incorporates the one-to-many interaction between agents has a greater expressiveness than its previous version that just contains one-to-one interactions. The quotient between expressiveness measures shows this improvement: the new version of the ML just needs one interaction for this many-to-many interaction where the old one needs several one-to-many interactions and, therefore, the new one needs less elements for representing the same models.

Even in the case of MLs (or versions of the same ML) comparison, the *quality of the modeling* (i.e. of the model for the solution system) can influence the number of modeling elements. For this reason, the presented framework strongly recommends that the same designers model the system in both MLs. In this manner, the quality of modeling is assumed to be similar in the two compared MLs. In the same line, it is also advised to use an heterogeneous group of several MAS designers, including experts of both MLs, to get an average modeling behavior. Another issue is that the proposed system to solve can introduce some bias in the measure, because the problem is best suited for one of the MLs. To avoid this, a battery of MAS for several problem domains can be established for the comparison of MAS MLs. This battery would constitute a benchmark. With this benchmark, a ML can be selected as the one with the expressiveness *unit*. Then, the expressiveness of any ML can be measured by comparison with the mentioned *unit* ML.

3 Measuring MAS MLs with the Availability and Specificity Metrics

This section introduces an account on the application of the availability and specificity metrics to compare results across different MLs. The comparison uses case studies from diverse MAS MLs. In order to gather data about availability and specificity from designers worldwide, our research group provides a spreadsheet (see Figure 1) that can be downloaded at `http://grasia.fdi.ucm.es/gschool`. In this spreadsheet, the designer ticks the necessary and missing concepts of a given ML (e.g. Tropos [3] in the figure) for a particular MAS example (e.g. *Cinema* in the figure). A value of "1" indicates that the corresponding modeling-concept of the ML is necessary for the considered problem domain,



Measuring Tropos

Concept \ Example	Cinema	Request	Delphi	Crisis	Deicing
. . .					
tasks	1	1	1	1	1
plan	1				1
. . .					
Availability	0,750	0,917	0,727	0,636	0,737
Specificity	0,750	0,458	0,667	0,583	0,583

Fig. 1. Spreadsheet for assisting in measuring Availability and Specificity metrics

while the blank space indicates the opposite. The most common missing concepts in each MAS ML are provided, but the user can add more if necessary. With this information, the spreadsheet automatically calculates the ratios to obtain the measurement of the metrics.

The MLs considered for this study are: Tropos [27]; PASSI and Agile PASSI [4]; Prometheus [20]; MASE [7, 8]; and INGENIAS [1]. The cited references describe the metamodels of these MLs and have been used for the computation of the metrics. These MAS MLs offer a wide perspective of the current state-of-the-art in Agent-Oriented Software Engineering.

The battery of examples used as problem domains for these measures includes the following case studies. The *Cinema* and *Request* examples are distributed with the IDK 2.7 [1], which is the support tool of the INGENIAS methodology. The first one provides support for users that want to get a ticket to see a movie. The second case study is a proof on how to construct a GUI connected to an agent. The *Delphi* example [10] illustrates how to reach consensus among experts evaluating the relevance of a document and it is already modeled and implemented with INGENIAS (demo at the conference AAMAS-2008). The *Crisis* management example uses a MAS to coordinate the mutual help of citizens facing a medical emergency in their city [17, 22]. A solution for this problem is distributed with the IDK 2.8 [1]. The remaining cases of the study have not been originally modeled with any of the MLs considered here. The *Deicing* [15] example considers the planning of the deicing and anti-icing processes for aircrafts in an airport. [15] formally models and solves this problem using constraints. The *Combat* [30] case study simulates combat systems with MAS. This example is taken from the simulation field. The goal of the *Steel* [12] case is the planning and observation of steel production, which comprehends several time-critical and highly interference susceptible processes. The *Box* [6] case study simulates a box manufacturer, which pursues the just-in-time-delivery of its products with high standards of quality. Finally, the *Scheduler* [24] example schedules large transportation networks as wholes, with a MAS to distribute the scheduling.

Table 1. Measuring Availability and Specificity of several MAS MLs

	Tropos		PASSI		Agile-PASSI	
	availability	specificity	availability	specificity	availability	specificity
Cinema	75.0%	75.0%	88.2%	75.0%	68.7%	73.3%
Request	91.7%	45.8%	90.1%	50.0%	80.0%	53.3%
Delphi	72.7%	66.7%	88.9%	80.0%	70.6%	80.0%
Crisis	63.6%	58.3%	85.7%	90.0%	75.0%	100.0%
Deicing	73.7%	58.3%	80.0%	80.0%	75.0 %	100.0%
Combat	53.8%	29.3%	64.3%	45.0%	53.8%	46.7%
Steel	70.0%	58.3%	72.2 %	65.0%	55.6%	66.7%
Box	53.3%	33.3%	68.8%	68.8%	57.1%	53.3%
Scheduler	60.0%	50.0%	78.3%	90.0%	56.5%	86.7%
Average	68.2%	52.8%	79.7%	70.0%	65.8%	73.3%

	Prometheus		MaSE		INGENIAS	
	availability	specificity	availability	specificity	availability	specificity
Cinema	85.7%	94.7%	77.7%	60.9%	100.0%	25.7%
Request	90.0%	47.4%	100.0%	52.2%	100.0%	14.9%
Delphi	84.2%	84.2%	82.4%	60.9%	97.3%	24.3%
Crisis	72.7%	84.2%	77.7%	60.9%	97.3%	25.0%
Deicing	86.7%	68.4%	76.5%	56.5%	97.4%	25.7%
Combat	85.7%	63.2%	68.8%	47.8%	92.1%	23.6%
Steel	76.5%	68.4%	75.0 %	65.2%	92.7%	25.7%
Box	84.6%	57.9%	68.8%	47.8%	94.9%	25.0%
Scheduler	73.9%	89.5%	73.9%	73.9%	90.2%	25.0%
Average	82.2%	73.1%	77.9%	58.5%	95.8%	23.9%

Table 1 presents the results for the availability and specificity metrics obtained with the previous MAS MLs (i.e. columns in the table) and case studies (i.e. rows in the table). Readers should remember that these metrics are based exclusively on the meta-model elements. So, they do not require a complete modeling but just the assessment of the usefulness of a certain concept in a given problem domain, which is characterized in this report by a case study.

Inspecting Table 1, one can observe that the specificity measurements regards on the complexity and the required level of detail in modeling for the problem domains. For instance, the Request and Box cases are simple and their related specificity measurements are low. By the same token, the Crisis-management and Deicing problem domains need a more complex modeling design and they get higher specificity-measurement values. In general, the higher level of detail a problem domain needs, the higher value it obtains in the specificity measurements. The reason is that this level of modeling detail directly influences the number of necessary modeling concepts of a metamodel (*ncmm* term) used in the definition of the specificity metric (see equation 3).

Regarding the availability measures, Table 1 shows that they depend on the features of MASs required to model each problem domain and which of them the ML provide. For instance, the Box and Scheduler problem domains primarily needs concepts for the individual features of agents, such as mental states and outputs of the agents; whereas the Request and Cinema problem domains needs more modeling concepts for interaction features. Since Tropos and Agile-PASSI focus more on interaction-agents features than on individual-agent features, they get low results in the Box and Scheduler problem domains (Tropos gets 53,3% for the Box and 60% for the Scheduler, and Agile-PASSI 57,1% for the Box and 56,5% for the Scheduler), while a high availability in the Request and Cinema cases (Tropos gets 91,7% for the Request and 75% for the Cinema, and Agile-PASSI 80% for the Request and 68,7%). According to the same reasoning, Prometheus gets better results in the availability metric for the Box and the Scheduler problem domains (84,6% and 73,9% respectively), because it includes a large group of modeling concepts for individual-agent issues. The reason for this dependence between availability-measures and the features of the problem domains can be concluded from the definition of the availability metric (see Equation 2). The availability measurements inversely depend on the number of missing concepts (mc term), and the mc term quantifies the absence of the modeling concepts for expressing the necessary features of MASs in a particular problem domain.

Furthermore, both for the availability and the specificity metrics, it can be regarded that high scores of the metrics correspond to a good adjustment of the ML to the scope of the domain problem. For instance, Prometheus is especially useful for BDI-like agents. Thus, its scores depend on whether the examples are BDI-like or not. This explains the lower scores (76,5% availability and 68,4% specificity) obtained in the Steel example, which is not BDI-like, and the higher scores in BDI-like developments like the Delphi (84,2% availability and 84,2% specificity) or the Cinema (85,7% availability and 94,7% specificity) examples.

Finally, some global observations can be done aggregating the results for the specificity (see Figure 2)) and availability (see Figure 3) metrics by methodologies Agile-PASSI is a subset of PASSI that only includes the most necessary concepts of PASSI, according to their authors. This explains the greater specificity of Agile-PASSI (see Figure 2), as the number of Agile-PASSI concepts is

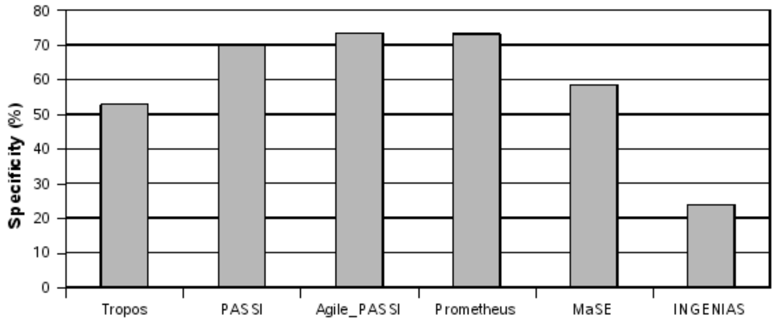


Fig. 2. Average specificity for each MAS Modeling Language

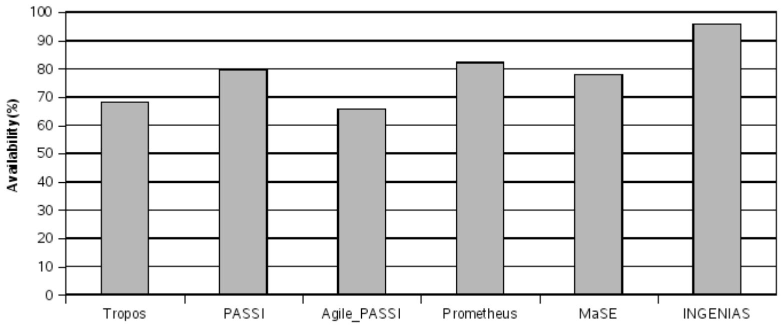


Fig. 3. Average availability for each MAS Modeling Language

smaller. However, this reduction also implies that the availability of Agile-PASSI decreases when compared with that of PASSI (see Figure 3), since some domain problems need concepts that appear at PASSI but not at Agile-PASSI. Another relevant case is INGENIAS. It is the methodology in the report with the highest score in availability (over 95%) due to its rich set of concepts. Nevertheless, this also makes many of its concepts not necessary for a given domain problem, what implies the lowest score in specificity (near 25%).

4 Analyzing the Improvements in a ML with the Expressiveness Metric

Section 2.4 discusses the use of the expressiveness metric for the comparison of the expressive power of versions of the same ML. This comparison relies on the count of model elements needed to solve a given problem. This section measures the improvements of the INGENIAS ML [1] between the versions 2.6 and 2.7. These versions are supported by the versions 2.6 and 2.7 of the IDK tool respectively.

The 2.7 version introduce support for one-to-many and many-to-many interactions. These interactions were not supported in the old version (i.e. 2.6). Thus, a MAS with one-to-many or many-to-many interactions can be represented directly with the new version, while in the old version these kind of interactions were represented with several one-to-one interactions. Thus, the new ML version is expected to have a grater expressiveness than the old one, as it needs less elements to represent the same MAS model.

The formal comparison between these two versions of the ML uses the *Request* MAS example already cited in the Section 3. In this MAS, an agent called *PersonalAssistant* performs a request for proposals to other agents, which are called *providers*. There is an one-to-many interaction from the PersonalAssistant agent to the provider agents. This study considers several numbers of providers, because the expressiveness ratio between both versions increases when considering a high number of providers.

Table 2. Comparison of Expressiveness between 2.7 and 2.6 versions, with the *Request* MAS example. The ratio of expressiveness is calculated with the ratio of numbers of elements (see Equation 5). The number of elements is the addition of entities and relationships.

# Providers	ML Version	# Entities	# Relations	# Elements	Ratio Expressiv.
3	2.7	52	45	97	$1,19(+19\%)$
	2.6	58	57	115	
10	2.7	101	94	195	$1,41(+41\%)$
	2.6	128	148	276	
100	2.7	731	724	1455	$1,61(+61\%)$
	2.6	1028	1318	2346	
∞	2.7	$52 + 7 * n$	$45 + 7 * n$	$97 + 14 * n$	$1,64(+64\%)$
	2.6	$58 + 10 * n$	$57 + 13 * n$	$115 + 23 * n$	

Table 2 gathers the data of the evaluation in this case. It shows that the 2.7 version of the INGENIAS ML is 19% more expressive than the 2.6 version. This increment considers the MAS with three providers. However this increment is greater when considering higher numbers of providers. The increments are 41%, 61% and 64% for ten, one hundred and infinity providers respectively. For the measurement with an infinity number of providers, n represents the number of additional providers apart from the three initial ones. Thus, the expressiveness metric confirms a relevant improvement in the new version of the ML.

5 Related Work

The evaluation of ML for MAS has drawn a lot of attention in recent works. These researches share some common features. They usually focus on qualitative measures with Yes/No [28] or Bad/Fair/Good [5, 23, 26] scales, and even structured text [25]. Besides, the evaluated issues in many of them [25, 26, 28] are biased in their formulation to MASs like autonomy, interactions, or organizations. On the contrary, the evaluation framework proposed in this paper uses quantitative measurement of the MLs and although the reported examples are intended for MAS, the approach is applicable to other kinds of MLs. As a drawback, the presented framework only measures the abstract syntax and partly the semantics, but not the notation or ontology as other approaches do [5, 25, 28].

There are also some works concerned with the definition of metrics to evaluate MLs with some common features with ours. Mulyar et al. [16] measure clinical computer interpretable MLs. Their metric consider the degree of support of 43 control-flow patterns. That is, they take into account the availability of several concepts (i.e. the control-flow patterns). Thus, the availability metric of the presented framework is similar to this approach. However Mulyar's evaluation is completely manual and targeted to the specific domain of clinical assistance, while our work defines the metric on the metamodel (therefore suitable for a partially automated processing) and is applicable to general MLs.

The expressiveness of MLs has also been considered in some previous works [23, 25] as a feature to consider in the evaluation of MAS MLs. The works of Kargl [13] (with the *explicitness* metric) and Vépa [29] (with metrics on metamodels about size or inheritance reusability) can be considered in this line. Nevertheless, these works do not provide any metric for expressiveness [25] or it is not clear how to use them in comparison [13, 29], whereas our evaluation framework includes a metric for measuring and quantitatively comparing the expressiveness.

The semantic evaluation of the MLs for a problem domain is another aspect to consider here. There are several works related to the evaluation of MLs. One of the best known is the one Opdahl and Henderson-Sellers [19] using the Bunge-Wand-Weber (BWW) ontological approach. Opdahl and Henderson-Sellers apply BWW to analyze and evaluate the Unified Modeling Language (UML) [11] for representing concrete problem domains. That work considers the Wand and Weber *ontological discrepancies*, such as the *Construct overload, redundancy, excess* or *deficit*. The *OPEN Modeling Language* (OML) [18] has also been evaluated in terms of the BWW model.

The presented evaluation framework relies on the judgment of experts about concepts clearly stated in the MAS literature in order to evaluate the appropriateness of the ML for a problem domain. However, the approach of Opdahl and Henderson-Sellers could complement our framework in some cases.

6 Conclusions and Future Work

This paper presents an evaluation framework for MAS MLs focused on their abstract syntax and their semantics. Over these information, the framework defines three quantitative metrics, i.e. *availability*, *specificity*, and *expressiveness*. These metrics makes the framework a suitable analysis tool for several processes related with MLs. Firstly, the framework offers a mechanism for the selection of the appropriate ML for a particular problem domain using the *availability* and *specificity* metrics. Secondly, the framework assists the designer in selecting a suitable subset of a ML for a specific kind of problem domain again with the *availability* and *specificity*, but also with the *expressiveness* metric to compare the expressive power of the MLs. Finally, the framework aids to measure the improvements between version of MLs with the *expressiveness* metric.

The framework is built upon two types of measures about the MLs. The first type relies on the abstract syntax of the models. In this case, they are objective, quantitative, and automated measures. The second one is semantic measures about the *necessary* elements in the metamodel to solve a given problem domain or modeling elements to represent a system. These measures are also quantitative but they depend on the assessment of experts, so they introduce a subjective component in the evaluation. In order to reduce this subjectivity, the presented framework strongly recommends using a large number of experts in the evaluation of the MLs and making that the same experts evaluate all the different MLs. In this way, the evaluation gets an average measure and it is not biased to certain MLs.

The future work considers several open issues for the framework. On one side, a complete benchmark will be defined for the availability and specificity metrics. This benchmark is a battery of MASs classified into several problem domains. Another useful benchmark could be defined for the expressiveness metric for MAS. In this case, a ML is considered as the unit ML. Then, the expressiveness of a MAS ML can be measured by comparison with the unit ML.

Finally, a more challenging issue is the evaluation of the presented framework against human judgments to provide an assessment of the level in which our metrics meet the human expectations.

Acknowledgements

This work has been supported by the project "Methods and tools for agent-based modelling" supported by Spanish Council for Science and Technology with grant TIN2005-08501-C03-03, and by the grant for Research Group 910494 by the Region of Madrid (Comunidad de Madrid) and the Universidad Complutense Madrid.

References

1. INGENIAS Development Kit. (March 6, 2008), `http://ingenias.sourceforge.net/`
2. Bauer, B., Müller, J.P., Odell, J.J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 91–103. Springer, Heidelberg (2001)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
4. Chella, A., Cossentino, M., Sabatucci, L., Seidita, V.: Agile PASSI: An Agile Process for Designing Agents. International Journal of Computer Systems Science & Engineering. Special issue on Software Engineering for Multi-Agent Systems (May 2006)
5. Dam, K.H., Winikoff, M.: Comparing Agent-Oriented Methodologies. In: Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2003. LNCS, vol. 3030, pp. 78–93. Springer, Heidelberg (2004)
6. Darley, V., Sanders, D.: An agent-based model of a corrugated-box factory: the trade-off between finished goods stock and on-time-in-full delivery. In: Proceedings of the Fifth Workshop on Agent-Based Simulation (2004)
7. DeLoach, S., Wood, M.F., Sparkman, C.H.: Multiagent Systems Engineering. International Journal of Software Engineering and Knowledge Engineering 11(3), 231–258 (2001)
8. DeLoach, S.A.: Multiagent systems engineering of organization-based multiagent systems. ACM SIGSOFT Software Engineering Notes 30(4), 1–7 (2005)
9. Fuentes-Fernández, R., Gómez-Sanz, J.J., Pavón, J.: Model integration in agent-oriented development. International Journal of Agent-Oriented Software Engineering 1(1), 2–27 (2007)

10. García-Magariño, I., Pérez Agüera, J.R., Gómez-Sanz, J.J.: Reaching Consensus in a Multi-agent System. In: 6th International Workshop on Practical Applications on Agents and Multi-agent Systems, IWPAAMS 2007, Salamanca, Spain, November 12-13, pp. 349–358 (2007)
11. OMG (Object Management Group). UML, Unified Modeling Language Infrastructure, Version 2.0 (2005) (Retrieved July 30, 2008), `http://www.omg.org/spec/UML`
12. Jacobi, S., Madrigal-Mora, C., León-Soto, E., Fischer, K.: AgentSteel: an agent-based online system for the planning and observation of steel production. In: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 114–119 (2005)
13. Kargl, H., Strommer, M., Wimmer, M.: Measuring the Explicitness of Modeling Concepts in Metamodels. In: ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML 2006), Workshop on Model Size Metrics, Genova, Italy (October 2006)
14. Kingston, J., Macintosh, A.: Knowledge management through multi-perspective modelling: representing and distributing organizational memory. Knowledge-Based Systems 13(2-3), 121–131 (2000)
15. Mao, X., ter Mors, A., Roos, N., Witteveen, C.: Agent-based scheduling for aircraft deicing. In: Proceedings of 18th Belgium-Netherlands Conference on Artificial Intelligence, BNAIC 2006, Namur, Belgium (2006)
16. Mulyar, N., van der Aalst, W.M.P., Peleg, M.: A Pattern-based Analysis of Clinical Computer-interpretable Guideline Modeling Languages. Journal of the American Medical Informatics Association 14(6), 781 (2007)
17. Oomes, A.: Organization awareness in crisis management. In: Proc. ISCRAM 2004, pp. 63–68 (2004)
18. Opdahl, A.L., Henderson-Sellers, B.: Grounding the OML metamodel in ontology. The Journal of Systems & Software 57(2), 119–143 (2001)
19. Opdahl, A.L., Henderson-Sellers, B.: Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. Software and Systems Modeling 1(1), 43–67 (2002)
20. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In: Proceedings of the Third International Workshop on Agent Oriented Software Engineering, at AAMAS (2002)
21. Pavón, J., Gómez-Sanz, J.: Agent Oriented Software Engineering with INGENIAS. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS, vol. 2691, pp. 394–403. Springer, Heidelberg (2003)
22. Schraagen, J., Eikelboom, A., te Brake, G.: Experimental evaluation of a critical thinking tool to support decision making in crisis situations. In: Proceedings of the 2nd International Conference on Information Systems for Crisis Response and Management, Brussels, Belgium, April 18-20 (2005)
23. Shehory, O., Sturm, A.: Evaluation of modeling techniques for agent-based systems. In: Proceedings of the fifth international conference on Autonomous agents, pp. 624–631 (2001)
24. Skobelev, P., Glaschenko, A., Grachev, I., Inozemtsev, S.: MAGENTA technology case studies of magenta i-scheduler for road transportation. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems (2007)
25. Sturm, A., Shehory, O.: A Framework for Evaluating Agent-Oriented Methodologies. In: Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2003. LNCS, vol. 3030, pp. 94–109. Springer, Heidelberg (2004)

26. Sudeikat, J., Braubach, L., Pokahr, A., Lamersdorf, W.: Evaluation of Agent-Oriented Software Methodologies-Examination of the Gap Between Modeling and Platform. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 126–141. Springer, Heidelberg (2005)
27. Susi, A., Perini, A., Mylopoulos, J.: The Tropos Metamodel and its Use. Informatica 29(4), 401–408 (2005)
28. Tran, Q.N.N., Low, G., Williams, M.A., Wales, N.S., Mary, A.: A Preliminary Comparative Feature Analysis of Multi-agent Systems Development Methodologies. In: Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M. (eds.) AOIS 2004. LNCS, vol. 3508, pp. 157–168. Springer, Heidelberg (2005)
29. Vépa, É., Bézivin, J., Brunelière, H., Jouault, F.: Measuring Model Repositories. In: Proceedings of the Model Size Metrics Workshop at the MoDELS/UML 2006 conference, Genova, Italy (2006)
30. Yang, A., Abbass, H.A., Sarker, R.: Landscape dynamics in multi-agent simulation combat systems. In: Webb, G.I., Yu, X. (eds.) AI 2004. LNCS, vol. 3339, pp. 39–50. Springer, Heidelberg (2004)

A Unified Graphical Notation for AOSE*

Lin Padgham[1], Michael Winikoff[1], Scott DeLoach[2], and Massimo Cossentino[3]

[1] RMIT University, Australia
{lin.padgham,michael.winikoff}@rmit.edu.au
[2] Kansas State University, USA
sdeloach@ksu.edu
[3] ICAR-CNR, Italy
cossentino@pa.icar.cnr.it

Abstract. Over the last five years several agent system development methodologies have been proposed and developed, with a number of them becoming well established and used beyond the group developing them. They all deal with similar concepts, but the notations used differ substantially. In this work we develop a standardized graphical notation for four prominent agent development methodologies, using principles of graphical notation suggested by Rumbaugh. We briefly illustrate the graphical design views produced in the different methodologies, on a conference management system example, using the standardized notation. We then discuss some of the similarities and differences on the basis of the design artifacts produced - which are now much more readily comparable than previously. This is a first step in being able to readily incorporate steps from different methodologies, depending on the needs of the application. It also helps to make the material more readily accessible to a wider audience.

1 Introduction

In recent years, it has become accepted that in order to effectively develop agent systems, it is necessary to have methodologies and notations that deal specifically with agent concepts and agent design issues. As a result, over the last several years many *Agent Oriented Software Engineering* (AOSE) methodologies have been developed or proposed, with some of the most well known including Gaia [1], O-MaSE (based on the earlier MaSE) [2], Tropos [3], Prometheus [4] and PASSI [5].

Important aspects of mature methodologies include the particular tools and diagrams that are used to develop and capture the analysis and design of the system being developed [6]. While there are a number of similarities between different methodologies cited above, each has its own particular strengths and nuances. It is certainly conceivable that a developer would wish to incorporate aspects of different methodologies into

* We acknowledge the input of Paolo Giorgini in discussing and determining the notation presented, and AgentLink which organized the Technical Forum. Padgham and Winikoff acknowledge the support of The Australian Research Council and the Australian Department of Education, Science and Training, as well as Agent Oriented Software under grants LP0453486 and CG040014. Scott DeLoach acknowledges the support of the US National Science Foundation under Grant No. 0347545 and by the US Air Force Office of Scientific Research.

a development process. In fact, this is the vision of method engineering [7] where the goal is to mix and match the activities, tasks and techniques of various methodologies according to the needs of a particular project. However, it is currently difficult to compare or use the diagrams and techniques from different AOSE methodologies because each methodology uses its own concepts, notations, and techniques.

The vision of method engineering is generally achievable using mature technologies such as object orientation where the basic concepts (objects, classes, associations, inheritance, etc.) and notations (UML) are well understood and generally agreed upon [8]. As a result of this maturity and agreement in the object oriented community, there are several well known activities, tasks and techniques that can be applied in a number of ways on various projects. There are also several commercially available tools that can be used together or separately to support a variety of approaches to developing object oriented systems [9]. Duplicating the success of object orientation requires two key elements: a common notation and a common metamodel.

The goal of this paper is to take a first step toward the level of maturity evidenced in the object-oriented community. In this first step, the developers of a number of the most detailed and prominent AOSE methodologies have worked together to produce a common notation.[1] While this first step is modest, we believe that a shared graphical notation is a first step toward making AOSE methodological work applicable to industry consumers. This notation will be used in each of our future individual methodological work and will be integrated into existing and new tools supporting that work.

In progress toward the second key element, there has been work done attempting to define a common metamodel for multi-agent methods and techniques [11]. Some efforts have also been spent in the field of standardization within the FIPA organization. Two different technical committees (Modeling and Methodology) worked on that and results describing their points of view can be found in [12,13]. However, while basic agent-oriented concepts have some commonality, we are far from having community-wide consensus on the majority of agent and multi-agent concepts. Thus, the common metamodels tend to be overly complex and of limited practical usefulness. Even though our goal may be considered more limited, from a practical standpoint, it is at least equally important and can provide a good first step in reaching such a community-wide consensus on the most important agent concepts.

While this paper does identify common concepts that we all use or wish to include in our notational set, the precise definitions and ways they are used do differ somewhat from methodology to methodology. Until there is a community wide agreement on these concepts, we believe these differences should be allowed to exist.

In the rest of this paper we present the new notation, motivating the choices we have made, followed by an example of a conference management system where we illustrate the design diagrams that can be produced by the various methodologies using the new notation. We finish with a brief discussion of the importance of working together across research groups to provide an engineering methodology that is accessible to practitioners wishing to build complex agent systems.

[1] We also worked with Paolo Giorgini, considering the Tropos methodology in the development of the common notation. We did not use Gaia because it does not make use of graphical models. We also did not include less prominent AOSE methodologies such as [10] for the time being.

2 The Unified Graphical Notation

We begin this section by describing general criteria for developing (graphical) notations suitable for the analysis and design of complex software systems. The article by Rumbaugh [14], one of the developers of the widely-used UML notation, gives the following list of desiderata for developing notations. This list illustrates the trade-offs that must be made when different desired properties conflict.

1. Clear mapping of concepts to symbols
2. No overloading of symbols
3. Uniform mapping of concepts to symbols
4. Easy to draw by hand
5. Looks good when printed
6. Must fax and copy well using monochrome images
7. Consistent with past practice
8. Self consistent
9. Distinctions not too subtle
10. Users can remember it
11. Common cases appear simple
12. Suppressible details.

In the remainder of this section, we present the notation that we have developed and explain the rationale for our decisions. As is often the case, there are sometimes trade-offs, but we believe we have now developed a notation that satisfies desirable properties for usability, clarity, etc.

Our notation uses a common type of diagram where the entities of interest are depicted as nodes and distinctive shapes are used to differentiate different types of nodes. Figure 1 presents an overview of our proposed notation. Relationships between entities are depicted by links, which can be decorated with a label giving the link type

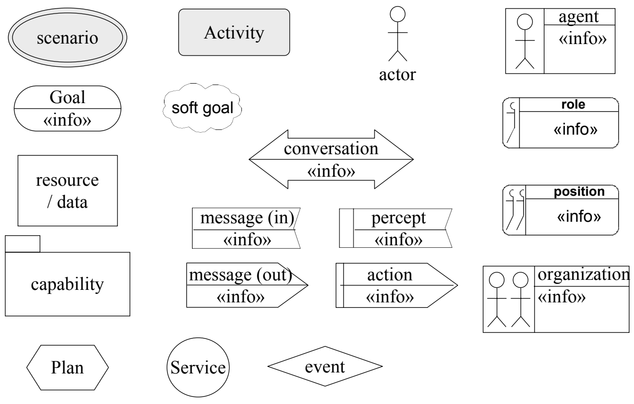


Fig. 1. Proposed Notation. The shaded symbols (use case and activity) as well as the actor symbol are existing UML symbols.

(e.g. "≪precedes≫", "≪initiates≫"). The decorations are optional and in many cases can be derived from the types of the entities. For example, in Prometheus an arrow from a percept to an agent is always a ≪receives≫ relationship.

This "graph-based" notation is standard in all types of engineering and is especially well suited to capturing system structure. However, capturing system *behavior* may be best done with non-graph-based models such as AUML sequence diagrams [15]. In this paper we do not tackle this type of diagram: since the AUML sequence diagram is well-defined and widely used, it makes little sense to propose a replacement for it. Other diagrams capturing system behavior, such as the Prometheus process diagrams, can be drawn with the proposed new notation.

Below we explain each type of node in our proposed notation. For each node we explain our reason for choosing the depiction given in Figure 1, and relate it to the concepts it can be used to represent. However, before describing the graphical notation, we briefly motivate our choice of concepts.

In selecting the concepts to be represented in our notation, we chose concepts that were required to model agent-based systems as indicated by their use in the four methodologies participating in the discussion, as well as other agent based methodologies of which we were aware. In identifying "required" concepts, we related the concepts used to design and build agent systems to the *defining properties* of agents [16][2]:

- Agents are *autonomous* – the key concept here is the notion of an *agent* itself, as an autonomous entity (distinct from objects).
- Agents are *situated* – the minimal key design concepts are the interface to the agent system's environment, in terms of *actions* performed by agents that affect the environment, and *percepts*[3], that get information from the environment. Clearly, more sophisticated concepts can be used to characterize the environment.
- Agents are *proactive* – the corresponding concept is *goals*.
- Agents are *reactive* – the corresponding concept is the notion of an *event*, a "significant occurrence".
- Agents are *social* – here a wide range of concepts could be used, ranging from the minimal one of messages, through to a range of organizational models. We choose to use the concepts of *messages*, *conversations*[4], *roles*, *positions* and *organizations*, where positions are placeholders for one or more roles within an organization, and an organization can include particular forms of organization, such as a team, or an e-institution.

In addition to these clearly required concepts, we added the following commonly used concepts:

- *Soft-goals*: goals that do not have a clear satisfiability definition, used in a number of methodologies, both agent-oriented and non-agent-oriented, for modelling non-functional requirements such as security, usability, flexibility. We include soft-goals since they are clearly useful, and since they fit in very well with agent-based design, where agents have goals.

[2] Sturm *et al.* [10] proposed a similar set of concepts, based on our earlier work [16].
[3] From the Latin *perceptum*, same root as the word "perceive".
[4] Also known as "protocols" or as "interaction protocols".

- *Actor*: an external entity, which can be human or software. This concept is useful in early analysis, and is well established in existing practice.
- *Capability*: a concept often used in discussing agents and implemented first by the JACK agent-oriented programming language [17] and subsequently adopted and extended by Jadex. Capabilities are a modularization construct for agents which can contain things such as plans, events, data, and sub-capabilities.
- *Plan*: sometimes termed tasks, plans are a key concept in BDI agent platforms, and in other plan-based implementation platforms. Hence, it is clearly important for (detailed) design to support plans.
- *Resource/data*: like any other software, agents normally need to store data in some form and/or use existing resources. For notation purposes we a use a single symbol to depict data or resources, without distinguishing between resources (e.g. a printer) and data, or between different data formats (objects, belief sets, relational databases).
- *Service*: the use of services are becoming very popular in information systems design using what are called service-based multi-agent systems. Although services are currently only well-defined in PASSI, we believe that this is a growing area and thus it is important to be able to depict existing services that will be used.

Having identified the concepts used to define agent systems, we now turn to considering how to graphically depict these concepts (see Figure 1) in order to more easily model agent system designs. According to the desiderata identified by Rumbaugh [14], each of the key concepts should be mapped to a distinct symbol satisfying the first two criteria ("Clear mapping of concepts to symbols" and "No overloading of symbols"). In addition, we strived to select symbols that emphasize similarities between related concepts (e.g. between outgoing messages and actions) whilst using clearly distinct symbols for concepts that are clearly dissimilar ("Uniform mapping of concepts to symbols"). For example, the symbols for a plan and for an agent are completely different. The symbols selected are also easily drawn by hand; our notation does not rely on shading, line thickness, or any other distinctions that are subtle, confusing, or that do not copy/fax well. The only distinction between symbol shapes that is somewhat subtle, the use of rounded corners in roles and positions, is reinforced by the use of a modification to the stick figure within the symbol. Further, as is discussed below, we have strived for consistency with past practice, where appropriate. In particular, we have used the UML notation where it made sense to do so. However, as will be seen in the following sections, many of the concepts used to engineer agent systems do not exist in UML, and in this case we believe that it is important to have new and clearly distinct symbols for concepts that are new and clearly distinct. Finally we describe a notational mechanism for achieving scalability by suppressing details.

Goal and Softgoal: Perhaps because goals are a new concept in agents, and one of the differences that clearly distinguish agents from objects, there is no consensus on how to depict them graphically. For instance, O-MaSE depicts goals as a rectangle with a number and a name, Tropos uses a fully rounded box (a "pill" or "lozenge" shape), and Prometheus uses an oval. Since one of our aims is to be compatible with existing standards, and since GRL[5] appears to be in the process of being standardized[6] we choose to use a pill/lozenge shape for goals.[7] For softgoals the standard is to use a cloud shape. Although this is not always easy to draw using tools, we cannot justify inventing a new symbol when a widely used symbol already exists for the concept.

Scenario: Scenarios are closely related to use cases, and hence we want a symbol that is close to the existing UML symbol for a use case (an oval). However, we also want to avoid overloading symbols, thus we have elected to use a double-lined oval.

Entities: Actors, Agents, Roles: Actors are a well-established concept with a well-established notation (the stick figure) which we adopt. For agents it is important to have a symbol that is distinct from the UML class symbol. However, despite the importance of the agent concept to AOSE, there is no consensus on its depiction: Prometheus uses a rectangle containing a stick figure, whereas Tropos uses a circle. For our notation, we propose that the Prometheus notation of the stick figure in a rectangle be adopted. Including the stick figure suggests a relationship with actors, and reinforces that, like humans, agents are active autonomous entities. Roles are an abstraction of agents and, in fact, are used in two ways within the AOSE community: as a social notion and as a component. In the *social* approach, agents are assigned to play roles within some organization. In the *component* approach, which has been used in both O-MaSE and Prometheus, agents are designed by grouping roles. To help define the role symbol, we adopted a general notational principle that states that when there are two concepts and one is an abstraction of the other, we use the same symbol for the abstracted concept, but with rounded corners. Thus, our proposed symbol for a role is the same as an agent but with rounded corners. To further emphasize that roles are abstract and are not complete agents, we embed a “half stick figure” instead of the full stick figure used in the agent symbol.

Intra-Agent: Plan/Task, Capability/Module: Plans (sometimes called tasks, e.g. in MESSAGE and Tropos) are depicted by a range of symbols. Using a similar reasoning to goals, we adopt the GRL/Tropos/*i** symbol: a hexagon. For capabilities we adopt UML’s package symbol, since capabilities are conceptually package-like: they contain other entities.

Events and Messages: In Prometheus events and messages are both depicted as envelopes, which are memorable, easy to recognize, and easy to draw by hand. However this notation has three problems: firstly it is confusing to draw intra-agent events as messages, secondly it is not clear from the symbol whether the message is incoming or outgoing, and thirdly, the envelope symbol is not consistent with related UML notation. Thus, we propose to adopt the UML notation for *send signal actions* and *accept event actions* to depict sending and receiving messages respectively. This notation allows us to distinguish incoming and outgoing messages and provides consistency with standard

[5] `http://www.cs.toronto.edu/km/GRL/`

[6] By the international telecommunications union (ITU). The proposed standardization brings together Use Case Maps (UCMs) and the Goal-Oriented Requirement Language (GRL) under the name User Requirements Notation (URN).

[7] GRL and Tropos use the same notation, due to their common ancestry, *i**.

practice. For events, we use a new symbol (a diamond). Events can be used to represent either inter- or intra-agent events.

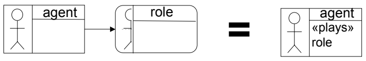
Environment: Percepts, Actions, Resource: Because we assume agents are situated in their environment, it can be argued that sending and receiving messages are actually special instances of the general notions of performing actions on the environment and receiving percepts from the environment. Therefore, we choose to use symbols for actions and percepts that are variants of the message symbols described above. The addition of a vertical bar as the distinguishing characteristic is somewhat arbitrary, but was chosen to be an obvious difference that is easy to draw. Resources are represented as simple rectangles in Tropos. Due to its simplicity and clarity, it makes sense to use this notation in the general sense as well. In addition, the rectangle symbol is similar to UML classes, which are also rectangular in shape. Therefore, the resource symbol can be seen as a generalization of the UML class.

Social concepts: Conversation, Organization, Position: Due to its mnemonic value, the Prometheus symbol for a protocol (a large double headed arrow, denoting bi-directional communication), is proposed for our notation. However, to clarify the concept it represents, we term the concept a “conversation” rather than a “protocol”. Because an organization is generally associated with a group of agents, it seemed natural to modify the agent symbol to represent this grouping. Thus, the agent symbol (a box with a single stick figure) is modified by replacing the single stick figure with multiple stick figures to represent an organization. When we decided on the symbol for a position, which is an organization’s place holder for a role (one or more), we used the organisation symbol, modified in the same way as the role symbol was modified from the agent symbol: a round-cornered rectangle with half stick figures.

Service: Since there is no accepted symbol for a service we propose a new symbol. In addition, since the concept of a service is not closely related to any of the other agent concepts discussed so far, we wanted a distinct symbol that is simple to draw. Thus, we propose using a simple circle to represent services. While the choice is somewhat arbitrary, it can be argued that a service is similar to a UML *interface* as it describes how to interact with the agent providing the service.

Links: Although some notations, such as Tropos and *i**, use a wide range of different link/arrow types, we do not believe this to be a good idea because they can be hard to draw by hand, are quite subtly different, and can be difficult for users to remember. Instead, we propose a single arrow type which is (optionally) enriched with textual annotations where desired to indicate different link types. We have identified a variety of useful links; however, this (partial) list can be easily extended as long as the meaning of the link is defined. Annotations include: a role ≪achieves≫ a goal, an event ≪occurs≫ during the pursuit of a goal, an event ≪triggers≫ the creation of a new goal, and a goal ≪precedes≫ another goal.

Scalability: Collapsing Links: For a design notation to be practical and usable for the design of large systems it must *scale* to large designs. There are a number of abstraction and packaging concepts in the notation that support this (such as organisation, protocol and capability). In addition we propose the use of “collapsible” links. As is shown below, many of the symbols have an information section, which can be used to indicate links with other entities such as an agent that ≪plays≫ a role. These collapsable links can be used to replace links to symbols. For example, below the left part of the figure, showing an agent with a link to a role symbol, is equivalent to the right side of the figure where the role symbol has been removed and the relationship indicated in the agent's information section.



The design notation can also be used to develop models which capture different aspects of the system, partitioning or abstracting to obtain scalability. For example a system overview diagram shows no agent internals, whereas agent overview diagrams can partition the system into a set of separate diagrams, one for each agent type.

3 Using the Notation

To illustrate the use of the unified notation across the different methodologies, we present examples of various diagrams taken from our methodologies based on a common exemplar system. Space limitations preclude us from presenting a wide range of diagrams from each methodology and we hope that the diagrams included are sufficient to give some of the flavour of how the proposed notation would be used.

The example we use in this paper is the popular multiagent conference management system, which was first proposed by [18] in 1998. It has since been widely used as it is suitable for illustrating a wide variety of aspects of multi-agent system analysis and design. The version of the system we are following is based on the version used in [19].

The conference management system is a multiagent system that supports the management of conferences that require the coordination of several individuals and groups to handle the paper selection process. This process includes paper submission, paper reviews, paper selection, author notification, final paper collection, and the printing of the proceedings. Authors may submit papers to the system up until the submission deadline. Once the submission deadline has passed, members of the program committee (PC) review the papers by either contacting referees and asking them to review a number of the papers, or by reviewing them themselves. Once all the reviews are complete, a final decision is made on whether to accept or reject each paper. Each author is notified of this decision and authors with accepted papers are asked to produce a final version that must be submitted to the system. All final copies are collected and sent to the printer for publication in the conference proceedings.

In the remainder of this section, we present several of the models used in our methodologies to capture various aspects of the conference management system analysis and design. However, each of the models uses the unified notation to illustrate how the different models might possibly be used together even though they are from different methodologies.

The Prometheus Goal Overview Diagram, as shown in Figure 2, shows how the overall goal of the system is refined into a hierarchical goal tree where subgoals define

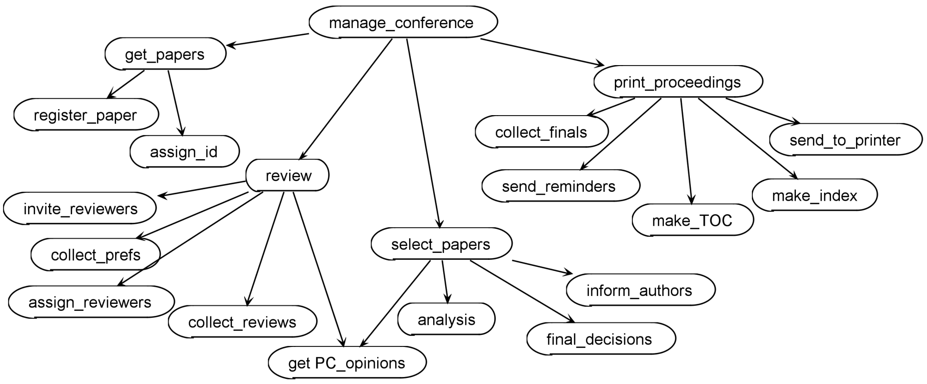


Fig. 2. Prometheus Goal Overview Diagram

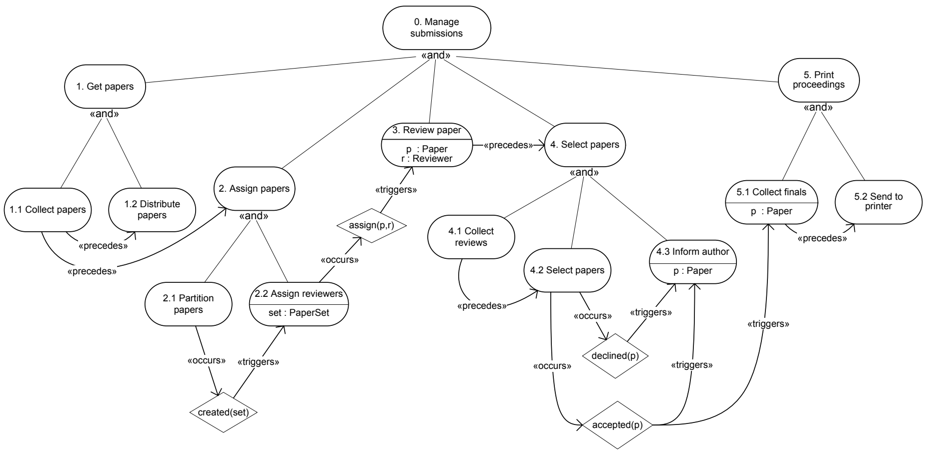


Fig. 3. O-MaSE Goal Model

how their parent goal may be achieved. In this case, the overall goal *manage_conference* is refined into four subgoals: *get_papers*, *review*, *select_papers*, and *print_proceedings*. Each of these goals is further refined into subgoals providing more insight into how the goals will be achieved.

The O-MaSE Goal Model shown in Figure 3 is similar to the Prometheus Goal Overview Diagram in function; however, it provides a richer set of constructs with which to model the goal structure. As in the Prometheus model, the O-MaSE Goal Model has a top level goal of *Manage Conference Submissions*, which is broken down into five conjunctive sub-goals: *Get Papers*, *Assign Papers*, *Review Paper*, *Select Papers*, and *Print Proceedings*. The "precedes" relation between the *Collect Papers* and *Distribute Papers* goals indicates that the *Collect Papers* goal must be achieved before work may begin towards the achievement of *Distribute Papers*. The "occurs" and

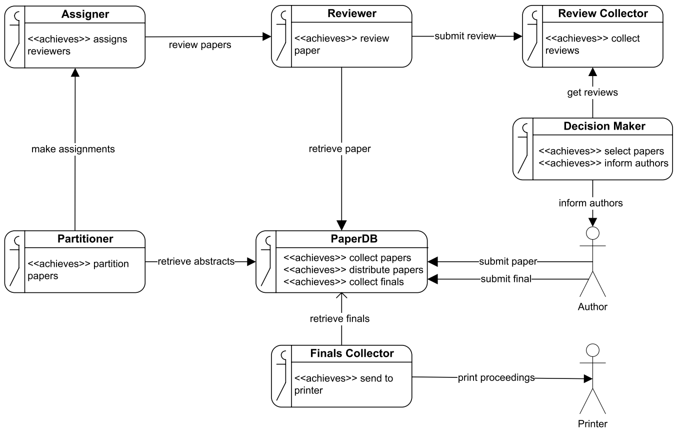

Assigner
<<achieves>> assigns reviewers
review papers
Reviewer
<<achieves>> review paper
submit review
Review Collector
<<achieves>> collect reviews
get reviews
Decision Maker
<<achieves>> select papers
<<achieves>> inform authors
make assignments
retrieve paper
inform authors
Partitioner
<<achieves>> partition papers
retrieve abstracts
PaperDB
<<achieves>> collect papers
<<achieves>> distribute papers
<<achieves>> collect finals
submit paper
submit final
Author
retrieve finals
Finals Collector
<<achieves>> send to printer
print proceedings
Printer

Fig. 4. O-MaSE Role Model

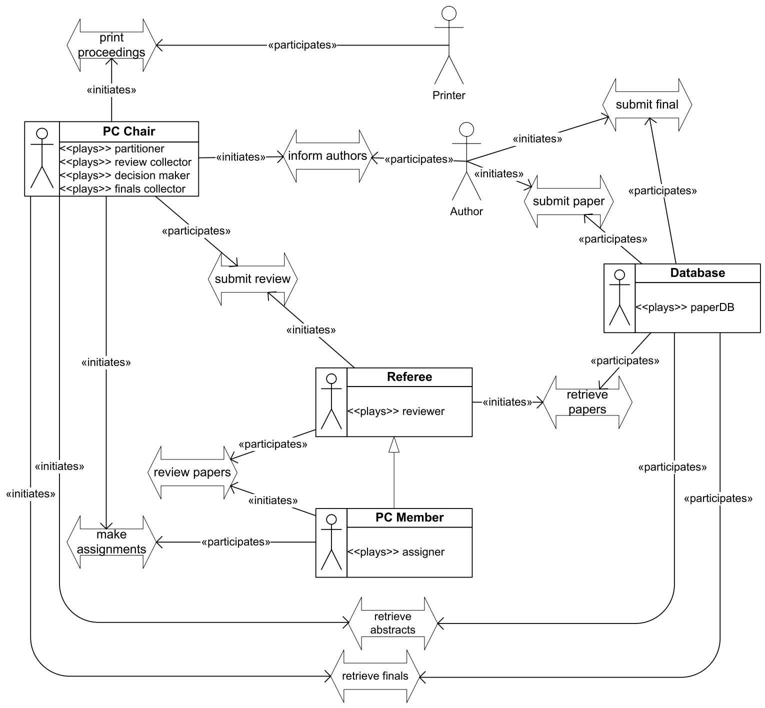


Fig. 5. O-MaSE Agent Model

"triggers" relation between the *Partition Papers* and *Assign Reviewers* goals and the *created(set)* event indicates that the *created(set)* event may occur during achievement of the *Partition Papers* goal and when it does, it triggers the creation of a new *Assign*

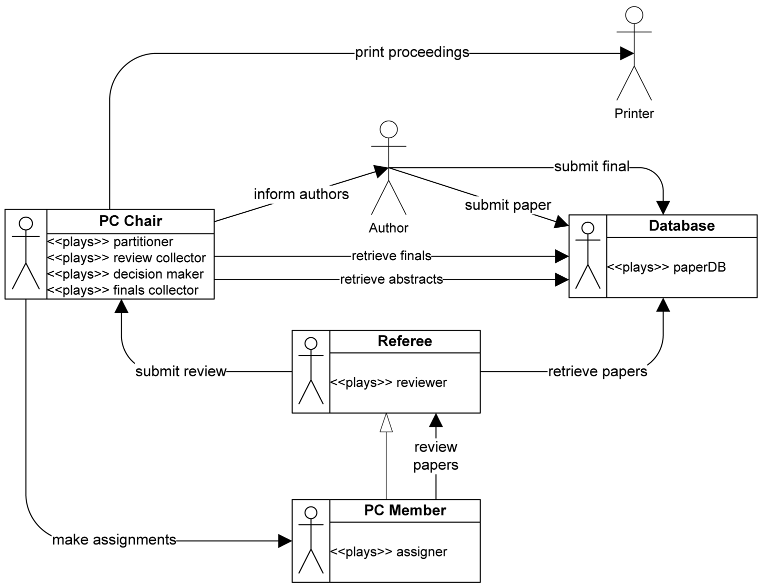


Fig. 6. O-MaSE Agent Model with Implicit Conversations

Reviewers goal that is parameterized based on some *set* of papers to be assigned to reviewers. As can be seen from Figures 2 and 3, the Prometheus Goal Overview Diagram provides a simpler and clearer model of system goals while the O-MaSE model provides additional constructs that provide a more detailed definition of system operation. Clearly, each model has situations where its use is warranted and the ability to choose between these models could be of great benefit to system designers.

The O-MaSE role model is derived from the goal model and depicts the relationships between the roles in the conference management system, as shown in Figure 4. In Figure 4, the goal(s) that each role may achieve are annotated via an embedded ≪achieves≫ relation in the body of each role symbol. Thus, the *Assigner* role is used to achieve the *Assigns Reviewers* goal. We also use a directed arrow to represent a conversation between roles with the arrows pointing from the initiator to the responder. The details of these conversations are defined using the commonly accepted AUML interaction diagrams [15]. Interactions with the external environment are represented as conversations with external actors.

The O-MaSE Agent Model (Figure 5) also shows assignment of roles to agents (via the ≪plays≫ embedded relation) but also shows the initiation and participation in specific conversations. (An alternate *implicit conversation* notation is shown in Figure 6). In both cases, the conversations between the agent types provide an overview of the entire system architecture.

The Prometheus System Overview Diagram (Figure 7) captures the architecture of the system, showing agent types, the conversations between them, and the interface to the environment in the form of percepts and actions. The System Overview diagram

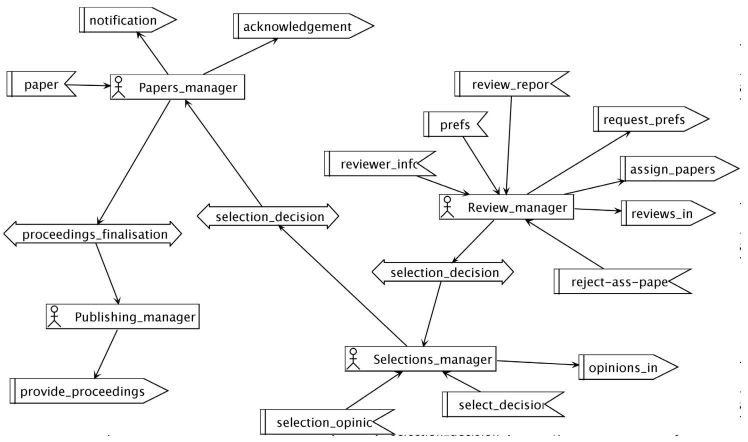


Fig. 7. Prometheus System Overview Diagram

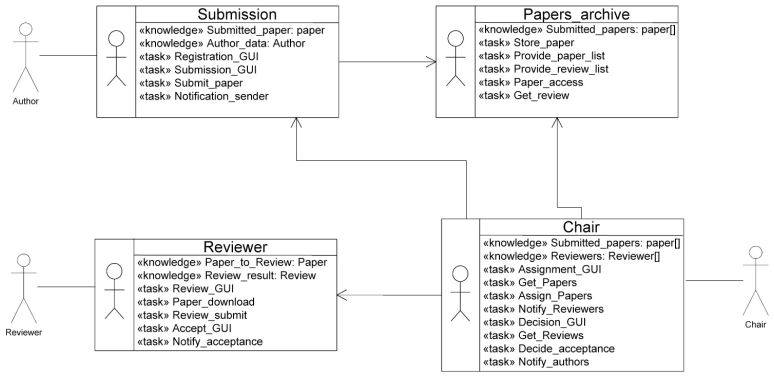


Fig. 8. PASSI Agent Structure Diagram

is generated automatically by the design tool (though layout must be done manually), based on the protocol specifications, and on the role specifications which form the agent.

The PASSI Multi-Agent Structure diagram (Figure 8) captures similar concepts as the O-MaSE Agent Model and the Prometheus System Overview. When following PASSI, the diagram contains no new information and is usually generated automatically by the PASSI ToolKit. Unique to the PASSI diagram is the use of ≪task≫ and ≪knowledge≫ keywords in the agent notation, which clearly highlights the extensibility of our current notation. While the O-MaSE Agent Model uses the ≪plays≫ keyword to denote the roles an agent may play, the PASSI approach is focused more on capturing the knowledge required by the agent (≪knowledge≫) and the tasks performed by the agents (≪task≫). Again, the commonality of the notation would allow the designer to use the most useful aspects of the various methods and diagram types to express the system design.

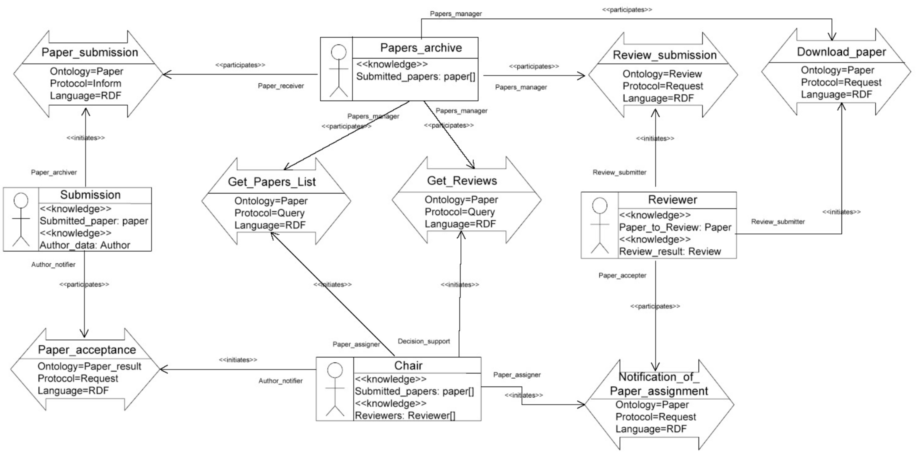


Fig. 9. PASSI Communication Ontology Description Diagram

The PASSI Communication Ontology Description diagram (Figure 9) is essentially composed of communications and agents. For each communication, the designer can introduce three parameters: the ontological elements exchanged in message contents (represented by the Ontology parameter), the agent interaction protocol (represented by the Protocol parameter), and the content language (represented by the Language parameter).

The Prometheus System Overview, the O-MaSE Agent Model and the PASSI Agent Structure and Communication Ontology Description diagrams all show conversations between agent types. The difference lies in how they represent interaction with the environment. The Prometheus models show an explicit representation of individual actions/percepts while the O-MaSE and PASSI models represent interactions via conversations with external actors.

4 Discussion and Future Work

We can see that once the gratuitous incompatibility of notation is removed, it becomes much easier to see both the similarities and the differences, and to consider extending one methodology with aspects of another. It is clear from the example and associated diagrams that O-MaSE and Prometheus are quite close, at least at the level of system specification and architectural design, whereas PASSI is more dissimilar:

– Both O-MaSE and Prometheus capture goals in a goal overview diagram. The Prometheus notation is simpler whereas the O-MaSE notation captures additional relationships, such as one goal triggering another.
– O-MaSE, Prometheus and PASSI all have a diagram that captures the roles in the system and in the case of Prometheus and O-MaSE these both indicate the assignment of goals to roles.

– The System Overview Diagram of Prometheus and the Agent Model of O-MaSE are virtually identical apart from O-MaSE showing actors, whereas Prometheus shows actions and percepts. PASSI on the other hand has a simpler Agent Structure diagram with a separate diagram for the communication ontology.

Although there is still some way to go before portions of the methodologies would be fully interchangeable, the unified notation does allow us to more readily see possibilities for borrowing from each other. Most importantly, the unified notation has the potential to allow users and developers to more readily understand the various methodologies and associated diagrams, as they do not need to learn a new 'language' for each approach.

In order to move towards this new unified notation, the authors are committed to using this notation, and to moving our respective CASE tools towards using this notation. Indeed, there already is a version of the Prometheus Design Tool that uses the new notation, and this was used to generate Figure 7.

In future work we hope to specify XML representations for certain diagrams, that will facilitate sufficient mapping between underlying models to allow some sharing of tools. We also hope that further collaboration and exploration can lead to further integration of our approaches to the benefit of industry developers wishing to use these technologies.

References

1. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: the Gaia methodology. ACM Transactions on Software Engineering and Methodology 12(3), 317–370 (2003)
2. DeLoach, S.A.: Engineering organization-based multiagent systems. In: Garcia, A., Choren, R., Lucena, C., Giorgini, P., Holvoet, T., Romanovsky, A. (eds.) SELMAS 2005. LNCS, vol. 3914, pp. 109–125. Springer, Heidelberg (2006)
3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An agent-oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems 8, 203–236 (2004)
4. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley and Sons, Chichester (2004)
5. Cossentino, M.: From requirements to code with the PASSI methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 79–106. Idea Group Inc., USA (2005)
6. Sturm, A., Shehory, O.: A framework for evaluating agent-oriented methodologies. In: Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2003. LNCS, vol. 3030, pp. 94–109. Springer, Heidelberg (2004)
7. Henderson-Sellers, B.: Method engineering for OO systems development. Commun. ACM 46(10), 73–78 (2003)
8. Bernon, C., Cossentino, M., Gleizes, M., Turci, P., Zambonelli, F.: A study of some multi-agent metamodels. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 62–77. Springer, Heidelberg (2005)
9. Object Management Group: UML Resource Page (2006), `http://www.uml.org/`
10. Sturm, A., Dori, D., Shehory, O.: Single-model method for specifying multi-agent systems. In: The Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS), pp. 121–128 (2003)

11. Bernon, C., Cossentino, M., Pavón, J.: Agent-oriented software engineering. Knowl. Eng. Rev. 20(2), 99–116 (2005)
12. Odell, J., Nodine, M., Levy, R.: A metamodel for agents, roles, and groups. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 78–92. Springer, Heidelberg (2005)
13. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardization to research. International Journal on Agent Oriented Software Engineering 1(1) (2007)
14. Rumbaugh, J.: Notation notes: Principles for choosing notation. Journal of Object Oriented Programming 9(2), 11–14 (1996)
15. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent UML. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 16–30. Springer, Heidelberg (2005)
16. Winikoff, M., Padgham, L., Harland, J.: Simplifying the development of intelligent agents. In: Stumptner, M., Corbett, D.R., Brooks, M. (eds.) Canadian AI 2001. LNCS (LNAI), vol. 2256, pp. 555–568. Springer, Heidelberg (2001)
17. Busetta, P., Howden, N., Rönnquist, R., Hodgson, A.: Structuring BDI agents in functional clusters. In: Jennings, N.R. (ed.) ATAL 1999. LNCS, vol. 1757, pp. 277–289. Springer, Heidelberg (2000)
18. Ciancarini, P., Niestrasz, O., Tolksdorf, R.: A case study in coordination: Conference Management on the Internet (1998), `ftp://cs.unibo.it/pub/cianca/coordina.ps.gz`
19. DeLoach, S.: Modeling organizational rules in the multi-agent systems engineering methodology. In: Cohen, R., Spencer, B. (eds.) Canadian AI 2002. LNCS, vol. 2338, pp. 1–15. Springer, Heidelberg (2002)

Prometheus and INGENIAS Agent Methodologies: A Complementary Approach

José M. Gascueña and Antonio Fernández-Caballero

Departamento de Sistemas Informáticos
Instituto de Investigación en Informática de Albacete (I3A)
Universidad de Castilla-La Mancha, 02071-Albacete, Spain
{jmanuel,caballer}@dsi.uclm.es

Abstract. A great number of methodologies to develop multi-agent systems (MAS) have been proposed in the last few years. But a unique methodology cannot be general enough to be useful for everyone without some level of customization. According to our knowledge, existent agent-based surveillance systems have been developed ad-hoc and no methodology has been followed. We are interested in creating tools that allow to model and to generate monitoring environments. This has motivated the selection of Prometheus and INGENIAS methodologies, to take advantage of both approaches in developing agent-based applications. In this paper a collection of equivalences between the concepts used in both methodologies is described extensively.

1 Introduction

The use of surveillance systems has grown exponentially during the last decade, and has been applied in many different environments [29]. A distributed configuration is mandatory to get scalable and robust surveillance applications ([28], [27], [18], [22]). These systems are complex and work in highly dynamic environments, where scattered sensors (e.g., camera, temperature, presence detection, and microphone) can decide and act with some degree of autonomy, and cooperate and coordinate for complete tracking of special situations. These characteristics are often cited as a rationale for adopting agent technology [31]. In fact, this technology has already been used in several surveillance systems (e.g. [26], [16], or [1]). According to our knowledge, existent agent-based surveillance systems have been developed ad-hoc and no methodology has been followed. Nevertheless, using a methodology allows to share the same terminology, annotation, models, and development process [2].

A great number of methodologies to develop multi-agent systems (MAS) have been proposed in the last few years (e.g. Gaia, Tropos, Message, MaSE). But a unique methodology cannot be general enough to be useful for everyone without some level of customization [4]. Habitually, techniques and tools proposed in different methodologies are combined.

2 Combining Prometheus and INGENIAS

In [10] several agent-oriented software development methodologies are presented, an evaluation and comparison of this methodologies is carried out by Tran and Low (chapter 12), and Henderson-Sellers introduce in chapter 13 a conceptual framework that enables reusing methodology fragments to create a specific methodology for each project faced. Some works on agent methodology integration are related to Tropos to INGENIAS mapping [7], or FIPA protocols specified in AUML transformed in equivalent INGENIAS models [8]. Also, the CAFnE toolkit [3] provides a framework that facilitates domain experts in making modifications to a deployed agent-based system without the assistance of agent programmers. Recently, a proposal for facilitating MAS complete life cycle through the Protégé-Prometheus approach has been presented [24].

In our case, we shall reuse parts of Prometheus and INGENIAS methodologies due to the following reasons. Once both methodologies have been studied, we have observed that the process followed in INGENIAS [17] during the analysis and design phases of MAS is very complex and difficult to follow, because is it not clear how the different models are being constructed along the phases, despite the documented general guidelines. In addition, it does not offer guidelines helping to determine which the elements that form the MAS are. It is solely the experience of the developer that determines its identification. On the contrary, Prometheus does offer these guidelines. These guidelines are also able to serve as a help to the experts in MAS development. They will be able to transmit their experience to other users explaining why and how they have obtained the different elements of the agent-based application. In addition, Prometheus is also useful because it explicitly considers agent perceptions and actions as modeling elements. INGENIAS (1) is currently focused to model-driven development (MDD) [18], (2) offers a process to generate code for any agent platform [8], and, (3) is supported by INGENIAS Development Kit (IDK) [17] that can be personalized for any application domain. These three characteristics are not offered by Prometheus. Nowadays the use of model-driven engineering (MDE) techniques along the software development life cycle is gaining more and more interest [23]. An MDD has important benefits in fundamental aspects such as productivity, portability, interoperability and maintenance. Therefore, in the MAS field, it seems quite useful to use a methodology such as INGENIAS, which supports this approach.

3 In-depth Comparison of Prometheus and INGENIAS

Prometheus [13] defines a proper detailed process to specify, implement and test/debug agent-oriented software systems. It offers a set of detailed guidelines that includes examples and heuristics, which help better understanding what is required in each step of the development. This process incorporates three phases. The *system specification* phase identifies the basic goals and functionalities of the system, develops the use case scenarios that illustrate the functioning of the system, and specifies which are the inputs (percepts) and outputs (actions). It obtains the scenarios diagram, goal overview diagram, and system roles diagram. The *architectural design* phase uses the outputs produced in the previous phase to determine the agent types that exist in the system and how they interact. It obtains the data coupling diagram, agent-role diagram, agent acquaintance diagram, and system overview diagram. The *detailed design* phase centers on developing the internal structure of each agent and how each agent will perform his tasks within the global system. It obtains agent overview and capability overview diagrams. Finally, Prometheus details how the entities obtained during the design are transformed into the concepts used in a specific agent-oriented programming language (JACK); this supposes, in principle, a loss of generality. The debugging mechanisms used in Prometheus are described extensively in [15]. The Prometheus methodology is supported by Prometheus Design Tool (PDT) [25].

On the other hand, the foundation of INGENIAS is the definition of MAS meta-model and a set of MDD tools oriented towards agents (model edition, verification, validation and transformation) integrated in INGENIAS Development Kit (IDK). The meta-model and IDK can be customized for a specific application domains. The meta-model describes the elements that enable modeling MAS from different points of view - agent, organization, environment, goals and tasks, and interaction [17]. The agent perspective considers the elements to specify the behavior of each agent. The organization perspective shows the system architecture. From a structural point of view, the organization is a set of entities with relationship of aggregation and inheritance. It defines a schema where agents, resources, tasks and goals may exist. Under this perspective, groups may be used to decompose the organization, plans, and workflows to establish the way the resources are assigned, which tasks are necessary to achieve a goal, and who has the responsibility of carrying them out. The environment perspective defines the agents' sensors and actuators, and identifies the system resources and

Table 1. Comparing Prometheus and INGENIAS

	Prometheus	**INGENIAS**
Proper development process	YES	NO: Based in the USDP (analysis and design phases). An agile process is suggested in [19]
General process to generate code from the models	NO: Only obtains code for JACK language	YES: Based in template definitions
Iterative development process	YES	YES
Model-driven development (MDD)	NO: Only proposes a correspondence between design models and JACK code	YES
Requirements capture	YES	YES
Meta-model	YES [6]	YES
Mechanisms to discover agents and interactions among agents	YES: Groups functionalities through cohesion and coupling criteria	NO
Agent model	BDI-like agents	Agents with mental states

Table 2. Comparing PDT and IDK

	PDT	**IDK**
Supported methodology	Prometheus	INGENIAS
Interface references the development process	YES: Diagrams are grouped in three levels according to the three Prometheus phases	NO: Possibility to create packets that correspond to the diverse phases of the process. Models of each phase are added to the corresponding packet
Mechanisms to prioritize parts of the project	YES: Three scope levels (essential, conditional and optional) [21]	NO
Code generation	YES: JACK $http://www.agent-software.com/$	YES: JADE $http://jade.tilab.com/$
Report generation of the MAS specification in HTML	YES	YES
Model fragmenting in various pieces	NO: For instance, only one diagram may be created to in order to gather all the objectives of the system	YES
Save a diagram as an image	YES: Format .png. The image resolution can be configured	YES: Formats .jpg, .svg, .png, wbmp, .bmp and .jpeg
Deployment diagrams	NO	YES
Agent communication	Defined in basis of messages and interaction protocols. Does not use a specific communication language. For JACK, there is a module compliant with FIPA [32]	Defined in accordance with communication acts of the agent communication language (ACL) proposed by FIPA $http://www.fipa.org/specs/fipa00061/$
Utility to simulate MAS specifications before generating the final code	NO	YES: Realized on the JADE platform. It is possible to manage interaction and tasks, and to inspect and modify the agents' mental states
Plans Executable as stand-alone	Textual description YES	Graphical description YES
Integration in Eclipse	YES: See [14]	YES: in IDK version 2.7

applications. The goals and tasks perspective describes the relations between tasks and goals. The interaction perspective describes how the coordination among the agents is produced. INGENIAS, at difference with Prometheus, does not define its own development process; rather it adopts the unified software development process (USDP) [11] as a guideline to define the steps necessary to develop the elements and diagrams of MAS during the analysis and design phases of the USDP. Moreover, INGENIAS facilitates a general process to transform the models generated during the design phase into executable code for all destination platforms. This general process is based in the definition of templates for each destination platform and procedures for extraction of information present in the models.

Both Prometheus and INGENIAS methodologies support the facility to capture requirements. In the Prometheus system specification phase, a version of KAOS is used to describe the system's goals [30] complemented with the description of scenarios that illustrate the operation of the system. In addition, in [5] guidelines appear to generate the artifacts of the Prometheus system specification from organizational models expressed in i*. In INGENIAS, requirements capture is performed by means of use case diagrams. Then, use cases are associated to system goals, and a goals analysis is performed to decompose them into easier ones, and finally tasks are associated to get the easiest goals.

In summary, INGENIAS has several advantages as opposed to Prometheus (see Table 1): (a) it follows an MDD approach, (b) it facilitates a general process to transform the models generated during the design phase into executable code. The advantages of Prometheus can be used (following the process to discover which be the agents of the system and its interactions) to enhance INGENIAS.

In Table 2 the Prometheus Design Tool (PDT) and INGENIAS Development Kit (IDK) tools are compared. It may be observed that PDT only has one advantage with respect to IDK: it has a mechanism to prioritize parts of a project. In the rest of considered characteristics, IDK equals or surpasses PDT. The tool that it will use to support the new methodology is IDK as it is independent from the development process and it may be personalized for the application under development.

4 Mapping Prometheus into INGENIAS

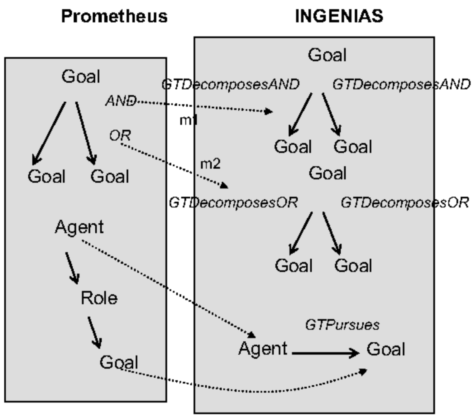
In the new methodology, we propose that the MAS developer firstly follows the system specification and architectural design phases proposed in Prometheus. Thus, an initial model in accordance with Prometheus is obtained using the guidelines that enable to identify the agents and their interactions. Afterwards, an equivalent model in INGENIAS is obtained using the collection of equivalences (mappings) between the concepts used in both methodologies. Next, modeling goes on with INGENIAS; thus, benefiting from the advantages of model-driven software development.

The detailed design phase of Prometheus is not followed because it mentions a specific agent-based model, namely the BDI model, which is different from the mental state agents used in INGENIAS. In this section, the collection of equivalences is described. We would like to point out that we are only describing the mapping from Prometheus into INGENIAS of the concepts used during the system specification and architectural design phases defined in Prometheus.

The proposed mappings have mainly been deduced on the basis of the organizations and relations between organizations that are possible to create with the supported tools PDT and IDK, respectively. In our next exposition, we use the Entity 1 - *RelationX* $\rightarrow$ Entity 2 notation to express that Entity 1 and Entity 2 are related through relation *RelationX*. The direction in which the arrow is pointing accurately reflects what the graphic representation of the relation is like. In the figures, dotted arrows are used to stand out how the organizations of Prometheus are transformed into the equivalent organizations of INGENIAS. In the tables information is offered on the models in which the structures represented graphically appear; and, textually, the transformations carried out are described.

4.1 Mapping Prometheus Goals

There are different approaches for the use of the term ”goal” [9]: (1) in classical planning, it is seen as a description of the state of the world to be



Prometheus	INGENIAS
Goal overview diagram	Goals and tasks model
Goal	Goal
AND dependences between goals	*GTDecomposesAND* relation
	GTDecomposesOR relation
OR dependencies	*GTDecomposes* if a goal has only one subgoal
Agent → Role → Goal	Organization model
	Agent model
	Goal and task model
	Agent → *GTPursues* → Goal

Fig. 1. Mapping information related with Prometheus goals into INGENIAS

reached - goals as aggregation; (2) in the BDI model, it is a wish to be satisfied - goals as entities; and, (3) to reflect the requirements that the system must fulfill in the design - goals as requirements. In INGENIAS, the goals are initially taken as self-representing entities (goals-as-entities approach) which guide the behavior of the agent. To take into account the planning approach (goals-as-aggregation approach), the goals must be allowed to connect with the set of elements (e.g. predicates) that they represent. As for the last approach (goals-as-requirements approach), the assimilation of goals with requirements is purely interpretative. It is up to the engineer to consider a goal as a requirement or not.

A goal that appears in the goal overview diagram used in Prometheus will correspond to a goal in the goals and tasks model in INGENIAS. *AND* and *OR* dependencies between goals can be established in both models; therefore, it is possible to directly transfer these relations from one model to another. In the goals and tasks model, a *GTDecomposeAND* relation and a *GTDecomposeOR* relation will be established to reflect an *AND* and *OR* relation between goals, respectively. The arrows m1 and m2 of Fig. 1 highlight the transformation of the structures between goals in Prometheus into the equivalent structures in INGENIAS - m1 and m2 show the transformation of AND and OR structures, respectively. When a goal has only one sub-goal, *GTDecomposes* is used.

In the system roles[1] diagram of Prometheus methodology, relations between goals and functionalities are established. The latter will be grouped to determine the types of agents in the system - the relation among agents and roles, Agent $\rightarrow$ Role, appear in the agent-role grouping diagram, whilst the relation among roles and goals, Role $\rightarrow$ Goal, appear in the system roles diagram. Therefore, implicitly, there is a relation between goals and agents. In INGENIAS, one of the consistency criteria of the goals and tasks meta-model expresses that *"the goal that appears in a goals and tasks model must appear in an agent model or in an organization model"* (criterion 2). Basing ourselves on the information from the system roles diagram and taking into account the previous comments, for each goal, the following relations[2] will be established in INGENIAS: (a) Agent - *GTPursues* $\rightarrow$ Goal in the organization model, (b) Agent - *GTPursues* $\rightarrow$ Goal in the agent model and (c) Agent - *GTPursues* $\rightarrow$ Goal in the tasks and goals model. All these equivalencies are summarized graphically and textually in Fig. 1.

4.2 Mapping Prometheus Agents

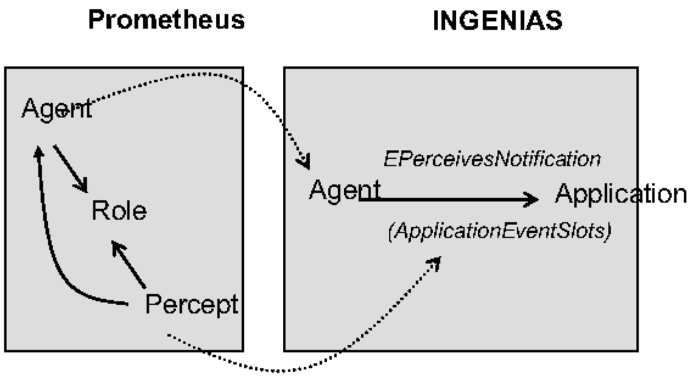
Every agent identified in the Prometheus methodology is reflected in INGENIAS in the agent model and in the organization model, based on the agent model consistency criterion *"for every agent in the organization model, there must be an instance for the agent model and vice-versa"* (criterion 4). If the agent must perceive changes in the environment, it will be also shown in the environment model. The agents that interact with other agents should also be represented in the interaction model. However, IDK only supports roles to generate the code that corresponds to an interaction. Therefore, for every agent identified in Prometheus, an associated role in the corresponding interaction model in INGENIAS to state its participation in the interaction has to be created. Likewise, we will establish an Agent - *WFPlays* $\rightarrow$ Role relation in the organization or agent model. Finally, we should remember that in the goals and tasks model, agents will be also obtained, according to what was specified in section 4.1.

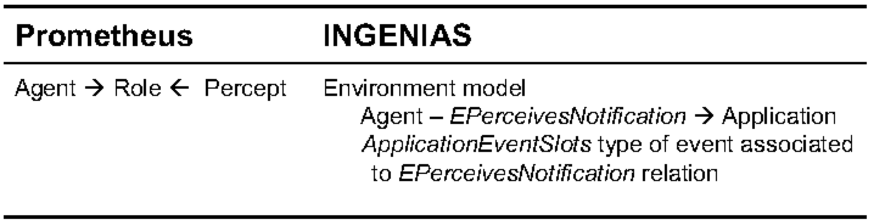
4.3 Mapping Prometheus Percepts and Actions

A percept is a piece of information from the environment received by means of a sensor. In Prometheus, percepts must at least belong to one functionality, and, thus, to the agent associated to that functionality, too. The relations among percepts and roles (Percept $\rightarrow$ Role) and the relations among percepts and agents (Percept $\rightarrow$ Agent) appear in the system roles diagram and the system overview diagram, respectively. The percepts of a Prometheus agent can be modeled in

[1] Sometimes, in the description of the mappings, the term role is used instead of the term functionality. Role is the term used in PDT, whereas functionality is the term used in Prometheus.

[2] What we really mean is that instances of the corresponding entities will be created and will be related through the pertinent relation.





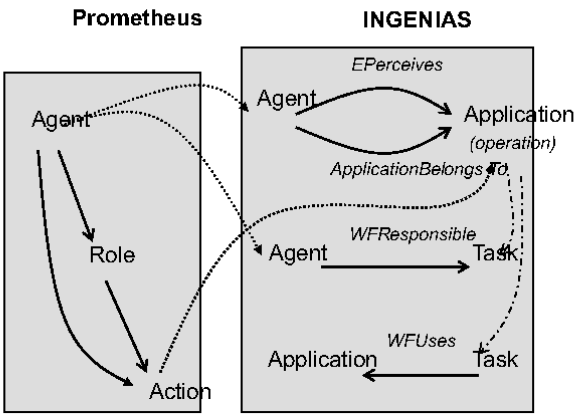
Prometheus	INGENIAS
Agent → Role ← Percept	Environment model Agent – *EPerceivesNotification* → Application *ApplicationEventSlots* type of event associated to *EPerceivesNotification* relation

Fig. 2. Mapping information related with Prometheus percepts into INGENIAS

INGENIAS by specifying a collection of operations in an application. Depending on whether the application existed prior to the MAS development or was developed ad-hoc, for this purpose, we can specify the application in an environment application or an internal application, represented by *EnvironmentApplication* and *InternalApplication*, respectively.

The consistency criterion 2 of the environment model states that *"every agent that perceives changes in the environment must appear in the environment model associated to an application"*. Therefore, in the environment model, an *EPerceivesNotification* relation between the agent and the corresponding application will be established. In a Prometheus percept descriptor, there is a field, Information carried, where it is specified the information transported as part of the percept. In INGENIAS, this information is included with an *ApplicationEventSlots* type of event associated to *EPerceivesNotification* relation. The basic ingredients of these equivalencies are graphically and textually summarized in Fig. 2.

In Prometheus, also every action must at least belong to one functionality and the agent associated to the functionality must execute it. The relations among actions and roles (Role → Action) and the relations among actions and agents (Agent → Action) appear in the system roles diagram and the system overview diagram, respectively. An action represents something that the agent does to interact with the environment. In INGENIAS, actions on the environment are assumed to be calls to operations defined in the applications. Therefore, an action



Prometheus	INGENIAS
	Environment model Agent – *EPerceives* → Application Agent – *ApplicationBelongs To* → *Application*
Agent → Role → Action	Agent model Agent – *WFResponsible* → Task Task – *WFUses* → Application
	Organization model Agent – *WFResponsible* → Task

Application can be refined in EnvironmentApplication or InternalApplication

Fig. 3. Mapping information related with Prometheus actions into INGENIAS

present in Prometheus will be transformed into an application operation present in the environment model in INGENIAS. *EPerceives* will be used to establish the relation between the agent and the application. In the environment model, an Agent - *ApplicationBelongs To* → Application relation will be also established to express that an agent uses an application. In addition, the operations in INGENIAS will be executed by the corresponding agent, by means of the relevant task. Thus, in the INGENIAS agent model, it will be created (1) an Agent - *WFResponsible* → Task relation to specify the agent responsible for carrying out the task, which triggers the execution of an action on the environment; and (2) a Task - *WFUses* → Application relation to express that an application is used in the task. If it is decided to specify the application in an environment or internal application, Task - *WFUses* → Environment Application and Task - *WFUses* → Internal Application is choosed, respectively. On the other hand, according to the INGENIAS agent model consistency criterion 1, *"every task associated to an agent must appear in the organization model, indicating its role within the task global structure"*, in the organization model, Agent - *WFResponsible* Task will appear. This information is summarized in Fig. 3.

4.4 Mapping Prometheus Data

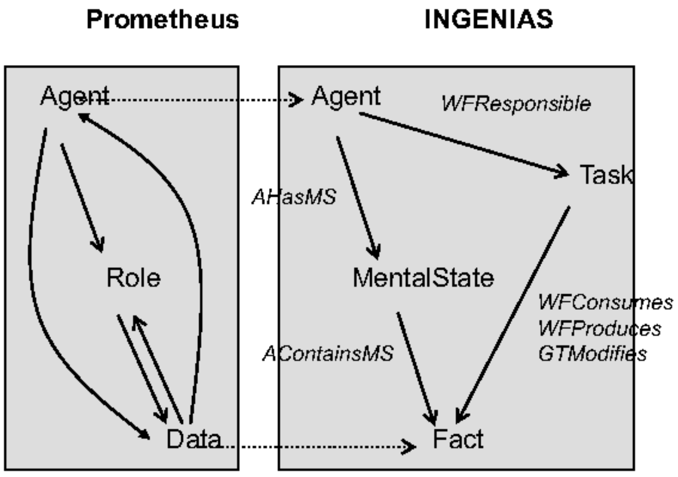
In INGENIAS, facts reflect information that is inherently true; for example, "water evaporates by applying heat", or any other information resulting from the execution of tasks. The data written and read by Prometheus agents will be made to correspond with facts or framefacts in INGENIAS. A framefact is a fact whose information is within its slots.

In the Prometheus system overview diagram, there are Agent → Data (expresses that the data is written by the agent) and Data → Agent (expresses that the data is read by the agent) relations. These structures would be translated to the INGENIAS agent model through the following procedure: a MentalState - *AContainsME* → Fact relation is created to specify a fact associated to a mental state, and an Agent - *AHasMS* → MentalState relation to specify that the mental state corresponds to the agent equivalent to the one we had in Prometheus. That is to say, an Agent - *AHasMS* → Mental State - *AContainsMS* → Fact structure will be get. The fact would become a Framefact instead of a Fact, if it includes more than one field of information. In INGENIAS, mental states are represented in terms of goals, tasks, facts, or any other entity that helps in state description. In INGENIAS goal and task model, Task - *WFConsumes* → Fact, Task - *WFProduces* → Fact, and Task - *GTModifies* → Fact relations are created to indicate that a fact is read, written and modified when executing a task, respectively. An agent will be responsible for executing that task - it is represented by an Agent - *WFResponsible* → Task relation. All these equivalencies are summarized graphically and textually in Fig. 4.

4.5 Mapping Prometheus Interaction Protocols

Prometheus offers a mechanism to derive interaction diagrams, and, as a result, interaction protocols from the scenarios developed. In one interaction protocol, there are agents that participate in the interaction and the messages that the agents send to one another.

In INGENIAS, the interaction model is used for describing how coordination between agents comes about. For each interaction protocol in Prometheus, we will develop an interaction model which will include: (1) an Interaction entity having the same name as the interaction protocol in Prometheus, (2) roles (from INGENIAS) associated to the agents that intervene in the interaction protocol, (3) an Interaction - *IInitiates* → Role relation, for the role associated to the agent that initiates the interaction protocol, (4) Interaction - *ICollaborates* → Role relations, for the roles associated to the agents (different to the initiating agent) that participate in the interaction protocol, and, (5) an Interaction - *IHasSpec* → GRASIASpecification relation. The GRASIASpecification entity will be associated to an interaction model where the messages sequence that intervenes in the interaction protocol, the agents that send them and their tasks involved will be described. In INGENIAS, the term Interaction Units, instead of messages, is used. In Prometheus, a protocol descriptor makes reference to the scenarios which include the protocol. When a scenario is created with PDT,



Prometheus	INGENIAS
Agent → Role ← → Data Agent ← → Data	Agent model MentalState – *AContainsMS* → Fact Agent – *AHasMS* → MentalState
	Goal and task model Agent – *WFResponsible* → Task Task – *WFConsumes* → Fact if Data is read Task – *WFProduces* → Fact if Data is written Task – *GTModifies* → Fact if Data is modified

Fig. 4. Mapping information related with Prometheus data into INGENIAS

a goal associated to the scenario is automatically generated. This is due to the contributions by Perepletchikov [21]. Therefore, the mentioned goal would be related to the interaction created in the INGENIAS interaction model.

5 Conclusions

After carrying out a comparative analysis of the Prometheus and INGENIAS methodologies, we realized that we can benefit from the simple guidelines offered by Prometheus in its development process to obtain an initial model of the MAS that we will be dealing with. Subsequently, we can move it into INGENIAS to proceed with the modeling, in order to benefit from the model-driven development. In order to make the transformations of concepts used in Prometheus into concepts used in INGENIAS the described mappings are applied.

At the moment these transformations are made manually. In order to implement our mapping proposal we plan to use some model transformation language. UML-AT [8] enables integrating N methodologies; but, as we are integrating only two, it is sufficient to use ATL (Atlas Transformation Language) [12]. In the foreseen implementation the INGENIAS Ecore meta-model will be used and the Prometheus Ecore meta-model will be created. Moreover, we are interested in going on adapting the IDK editor to model surveillance systems because it is independent from the development process and it may be personalized for the application under development.

Acknowledgements

This work is supported in part by the Junta de Comunidades de Castilla-La Mancha PBI06-0099 grant and the Spanish Ministerio de Educación y Ciencia TIN2007-67586-C02-02 grant.

References

1. Aguilar-Ponce, R., Kumar, A., Tecpanecatl-Xihuitl, J.L., Bayoumi, M.: A network of sensor-based framework for automated visual surveillance. Journal of Network and Computer Applications 30, 1244–1271 (2007)
2. Bordini, R.H., Dastani, M., Winikoff, M.: Current issues in multi-agent systems development. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 38–61. Springer, Heidelberg (2007)
3. Buddhinath Jayatilleke, G., Padgham, L., Winikoff, M.: A model driven development toolkit for domain experts to modify agent based systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 190–207. Springer, Heidelberg (2007)
4. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: From standardisation to research. International Journal of Agent-Oriented Software Engineering 1(1), 91–121 (2007)
5. Cysneiros, G., Zisman, A.: Refining Prometheus methodology with i*. In: 3rd International Workshop on Agent-Oriented Methodologies (2004)
6. Dam, K.H., Winikoff, M., Padgham, L.: An agent-oriented approach to change propagation in software evolution. In: Proceedings of the Australian Software Engineering Conference, pp. 309–318 (2006)
7. Fuentes, R., Gomez-Sanz, J.J., Pavón, J.: Integrating agent-oriented methodologies with UML-AT. In: Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1303–1310 (2006)
8. Fuentes, R., Gomez-Sanz, J.J., Pavón, J.: Model integration in agent-oriented development. International Journal of Agent-Oriented Software Engineering 1(1), 2–27 (2007)
9. Gómez Sanz, J.J.: Modelado de sistemas multiagente. Ph.D thesis, Departamento de Sistemas Informáticos y Programación. Universidad Complutense de Madrid (2002)
10. Henderson-Sellers, B., Giorgini, P.: Agent-Oriented Methodologies. Idea Group Publishing, USA (2005)

11. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Reading (1999)
12. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
13. Padgham, L., Winikoff, M.: Developing intelligent agents systems: A practical guide. John Wiley and Sons, Chichester (2004)
14. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the Prometheus methodology. In: First International Workshop on Integration of Software Engineering and Agent Technology, pp. 383–388 (2005)
15. Padgham, L., Winikoff, M., Poutakidis, D.: Adding debugging support to the Prometheus methodology. Engineering Applications of Artificial Intelligence 18(2), 173–190 (2005)
16. Patricio, M.A., Carbó, J., Pérez, O., García, J., Molina, J.M.: Multi-agent framework in visual sensor networks. EURASIP Journal on Advances in Signal Processing, Article ID 98639 (2007)
17. Pavón, J., Gomez-Sanz, J.J., Fuentes, R.: The INGENIAS methodology and tools. In: Agent-Oriented Methodologies. Idea Group Publishing, USA (2005)
18. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: Model driven development of multi-agent systems. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 284–298. Springer, Heidelberg (2006)
19. Pavón, J.: INGENIAS: Développement Dirigé par Modèles des Systémes Multi-Agents. Habilitation à diriger des recherches de l'Université Pierre et Marie Curie (2006)
20. Pavón, J., Gomez-Sanz, J.J., Fernández-Caballero, A., Valencia-Jiménez, J.J.: Development of intelligent multi-sensor surveillance systems with agents. Robotics and Autonomous Systems 55(12), 892–903 (2007)
21. Perepletchikov, M., Padgham, L.: Systematic incremental development of agent systems, using Prometheus. In: Fifth International Conference on Quality Software, pp. 413–418 (2005)
22. Remagnino, P., Shihab, A.I., Jones, G.A.: Distributed intelligence for multi-camera visual surveillance. Pattern Recognition 37(4), 675–689 (2004)
23. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2), 25–31 (2006)
24. Sokolova, M.V., Fernández-Caballero, A.: Facilitating MAS complete life cycle through the Protégé-Prometheus approach. In: Nguyen, N.T., Jo, G.S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS, vol. 4953, pp. 63–72. Springer, Heidelberg (2008)
25. Thangarajah, J., Padgham, L., Winikoff, M.: Prometheus Design Tool. In: Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 127–128 (2005)
26. Ukita, N., Matsuyama, T.: Real-time cooperative multi-target tracking by communicating active vision agents. Computer Vision and Image Understanding 97(2), 137–179 (2005)
27. Valencia-Jiménez, J.J., Fernández-Caballero, A.: Holonic multi-agent systems to integrate independent multi-sensor platforms in complex surveillance. In: IEEE International Conference on Advanced Video and Signal based Surveillance, vol. 49 (2006)
28. Valencia-Jiménez, J.J., Fernández-Caballero, A.: Holonic multi-agent system model for fuzzy automatic speech / speaker recognition. In: Nguyen, N.T., Jo, G.S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS, vol. 4953, pp. 73–82. Springer, Heidelberg (2008)

29. Valera, M., Velastin, S.A.: A review of the state-of-the-art in distributed surveillance systems. In: Intelligent Distributed Video Surveillance Systems. IEE Professional Applications of Computing Series, vol. 5, pp. 1–30 (2006)
30. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 249–263 (2001)
31. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review 10(2), 115–152 (1995)
32. Yoshimura, K.: FIPA JACK: A plugin for JACK Intelligent Agents^{TM}. Technical Report, RMIT University (2003)

The Formal Semantics of the Domain Specific Modeling Language for Multiagent Systems

Christian Hahn and Klaus Fischer

German Research Institute for Artificial Intelligence (DFKI)
Stuhlsatzenhasuweg 3
66123 Saarbrücken
{Christian.Hahn, Klaus.Fischer}@dfki.de

Abstract. Recently, associated with the increasing acceptance of agent-based computing as a novel computing paradigm a lot of research has been addressed to develop mechanisms and methods to support the agent-based development of complex software systems. Especially the idea to define agent-oriented languages on a more abstract level through metamodels is recently often applied. However, the metamodel's opportunity to express the language's semantics are restricted as only concepts and their relationships to each other can be defined within the metamodel. This paper discusses an approach to formalize the semantics of Dsml4mas—a modeling language for multiagent systems—to support the system designer in validating and verifying the generated design.

1 Introduction

Agent-oriented software engineering (AOSE) is a relative young field with its first workshop held in 2000. However, several methods and methodologies have been developed to facilitate the design of agent and multiagent systems like for instance modeling languages like Agent UML [1], methodologies like Tropos [2], or programming languages like JACK [3]. However as stated in [4], the field of verification and validation of multiagent systems (MASs) is comparatively less well-developed.

The development of agent-based software systems—similar to any other kind of software system—is mainly done in three phases: In a first step, the MAS is designed possibly adopting an existing AOSE methodology. In a second step, the resulting artifacts are taken as a base for manually programming the agent-based system and in a last step, the resulting system is debugged.

To close the gap between design and implementation, we developed a modeling language in the domain of multiagent systems (Dsml4mas) that provides key features like a concrete syntax, abstract syntax, and semantics as well as code generators that support the user in the three development steps.

For defining the abstract syntax, we defined a platform independent metamodel for MAS called PIM4Agents (see [5] for a detailed description) that is defined using UML[1] that provides a graphical notation to design object-oriented

[1] Unified Modeling Language: http://www.uml.org/

software systems. However, UML class diagrams are not sufficiently precise to set out all relevant aspects of a specification. Beyond straightforward constraints (e.g. association multiplicities) there exist a range of complex and sometimes subtle restrictions that are not easily conveyed in diagrammatical form. Additional information is needed in order to capture the semantics of a language which is important in order to give the language a clear representation and meaning. Otherwise, assumptions may be made about the language that lead to its incorrect use. Even if the developers may have an understanding of the syntax of a language, the semantics are the key to clarify the languages and concepts meanings. In terms of MDD, semantics are often introduced when transforming a platform independent model (PIM) to a specific platform that offers some kind of execution semantics. However, our aim is to introduce a clear semantics already at the PIM level to ensure that the generated models can already be validated on a more abstract level.

For specifying the semantics of Dsml4mas, we use the formal specification language Object-Z [6], which is a stated-based and object-oriented specification language. Object-Z is an extension of Z specification language [7,8] specialized on formalizing object-oriented specifications and bases on mathematical concepts (like sets, functions, and first-order predicate logic) that permits rigorous analysis and reasoning about the specifications.

The remainder of this paper is structured as follows: Section 2 briefly describes Object-Z and its foundations. Section 3 discusses parts of the abstract syntax of Dsml4mas and its corresponding semantics using Object-Z. Section 4 illustrates how the semantics are integrated in the provided modeling framework. Followed by Section 5 that states related work. Finally, Section 6 concludes this paper.

2 Object-Z Language

As aforementioned, Object-Z is an object-oriented specification language that supports features like classes, instance, inheritance, and polymorphisms.

The most important features of an Object-Z specification are class schemas (see Fig. 1) that take the form of a named box with optionally a list of generic parameters. Furthermore, a class schema includes (i) a list of visibility that restricts the access to variables and operations, (ii) a list of inherited classes, (iii) a list of variable definitions and invariants, as well as (iv) a list of operations.

The state schema in Object-Z consists of the set of declared variables and the corresponding class invariants. The operation schemas specify operations relating pre and post conditions of the object. Input variables are annotated by a question mark (?), output variables by an exclamation mark (!). A list marked with δ declares the set of variables that are changed by the operation.

Furthermore, Object-Z provides a set of operators that allow the combination of operations. This list of operators include the sequence operator ($op1 \fatsemi op2$), the conjunction operator ($op1 \wedge op2$), the choice operator ($op1 \mathbin{[]} op2$), the parallel operator ($op1 \parallel op2$), as well as the operation enrichment ($schema \bullet op$).

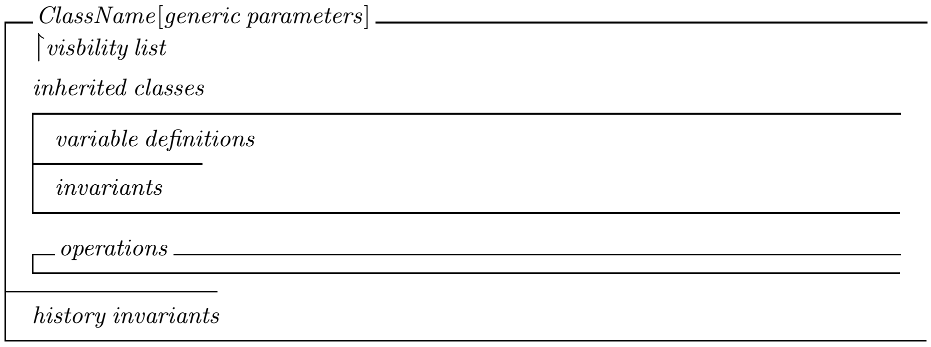


Fig. 1. A partial Object-Z class schema representation

Our approach for defining the semantics of Dsml4mas is to use the abstract syntax defined by the PIM4Agents metamodel as a base and transfer the related information to an Object-Z specification. Thus, the static semantics are defined by formalizing the concepts and relationships between the concept in the metamodel as attributes and invariants of the corresponding Object-Z class. Therefore, the UML's aggregation is mapped to the object aggregation of Object-Z expressed by Ⓢ, whereas the UML's composition is mapped to the object composition of Object-Z expressed by ⓒ. How to use this notation is illustrated in more detail in Section 3.

The denotational semantics are defined by introducing additional variables (we call these semantic variables to distinguish them from the variables that formalize the abstract syntax), which are used to define the semantics and invariants in Object-Z classes. Operational semantics are specified in terms of class operations and invariants restricting the operation sequences. We use the timed trace notation of the timed refinement calculus [9] to define these invariants with Object-Z. With this approach, we give a mutually consistent (formal) denotational and operational semantics.

The class invariants define the static semantics of Dsml4mas and thus define whether a model is meaningful. The class operations define the dynamic semantics and thus declare whether a model can be interpreted and executed.

3 Semantics of Dsml4mas

The PIM4Agents metamodel is divided into eight views (i.e. multiagent system, agent, organization, role, interaction, behavior, and environment) each emphasizing on a specific aspect of a MAS. In the following, we discuss selected aspects in more detail by specifying (i) the corresponding metamodel and (ii) the semantics of selected concepts using Object-Z. A more complete overview of PIM4Agents and its aspects can be found in [5] and [10].

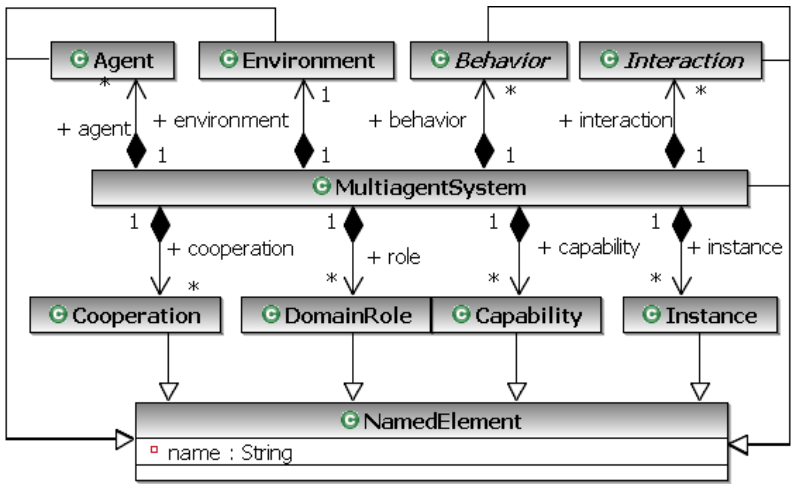


Fig. 2. The partial metamodel reflecting the multiagent system view of DSML4MAS

3.1 Multiagent System View

The partial metamodel of the *multiagent system view* is depicted in Fig. 2. The multiagent system aspect contains the main building blocks of a MAS and thus includes the concepts *MultiagentSystem*, *Agent*, *Instance*, *Cooperation*, *Capability*, *Interaction*, *DomainRole*, *Behavior*, and *Environment*.

The class schema below formalizes the definition of a *MultiagentSystem*. It inherits from the class schema of *NamedElement*. Its declarative part contains the variables *agent*, *instance*, *cooperation*, *capability*, *interaction*, *role*, and *behavior* that can be empty powersets (i.e. expressed by the symbol $\mathbb{P}$). Furthermore, each MultiagentSystem contains exactly one *environment*. The types of these variables correspond to the types defined in the multiagent system metamodel, i.e. the variable *agent* refers to the type *Agent*.

Critically, the set of *Agents* within the *MultiagentSystem*, i.e. *agents*, should be greater or equal to two (see [I 1]).

MultiagentSystem

NamedElement

$agent : \mathbb{P}\ Agent©,\ instance : \mathbb{P}\ Instance©,\ cooperation : \mathbb{P}\ Cooperation©$
$capability : \mathbb{P}\ Capability©,\ interaction : \mathbb{P}\ Interaction©$
$role : DomainRole©,\ behavior : \mathbb{P} \downarrow Behavior©$
$environment : Environment©$

$\#agent \geq 2$ [I 1]

3.2 Agent View

The *agent view* defines how to model single autonomous entities, the capabilities they have to solve tasks and the roles they play within the MAS. The metamodel of the agent view is depicted in Fig. 3. It includes the concepts *Agent*, *Instance*, and *Capability* as well as *Resource* (from the environment view), and *Behavior* (from the behavior view).

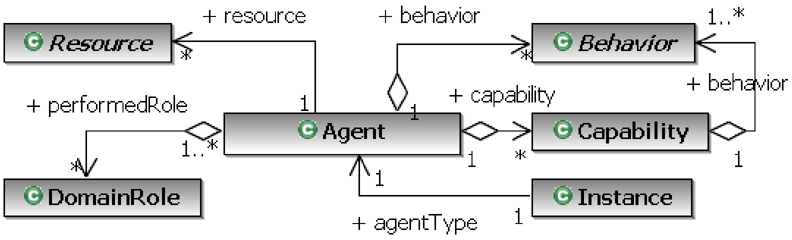


Fig. 3. The metamodel of the agent view

The semantics of the concept *Agent* is given in the class schema below. The class schema inherits from the *NamedElement*. Its declarative part consists of four variables, i.e. *performedRole*, *capability*, *behavior*, and *resource*. Furthermore, we introduce a semantic variable called *behaviorUsed* that unions all *Behaviors* an *Agent* is equipped with. This is done in [I 1] by defining the union of all *Behaviors* either directly used by the *Agent* or through the concepts *Capability* and *DomainRole*. To guarantee proactive and/or reactive behavior, invariant [I 2] ensures that an *Agent* needs to be equipped with at least one *Behavior* through testing if the variable *behaviorUsed* contains at least one element.

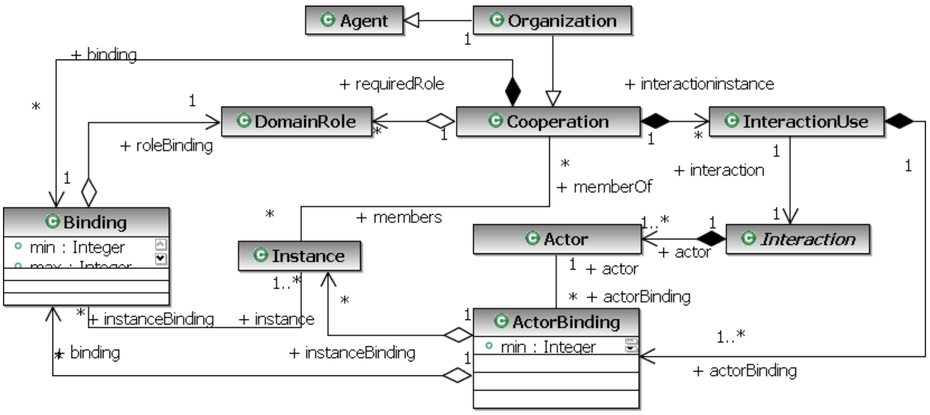


Fig. 4. The metamodel of the organization view

3.3 Organization View

The *organization view* (depicted in Fig. 4) defines how single autonomous agents are arranged to more complex social structures like for instance cooperations or organizations. The organization view includes the concepts *Cooperation*, *Organization*, *Binding*, *InteractionUse*, and *ActorBinging* as well as *Interaction* (from the interaction aspect), *Actor* and *DomainRole* (both from the role aspect), as well as *Agent*, and Instance (from the agent aspect).

Cooperation. The class schema of a *Cooperation* is defined below. It includes four variables (i.e. *members*, *interactioninstance*, *binding*, and *requiredRole*) and inherits from the class schema of *NamedElement* (see Fig. 2).

Cooperation

NamedElement

$interactioninstance : \mathbb{P}\, InteractionUse, members : \mathbb{P}\, Instance$
$requiredRole : \mathbb{P}\, DomainRole\circledS, binding : \mathbb{P}\, Binding\textcircled{c}$
Δ [Semantic Variables]
$instancesBoundToActor : \mathbb{P}\, Instance$

$\#members \geq 2$ [I 1]
$instancesBoundToActor = \bigcup\{ab : \bigcup\{ac : \{i : \bigcup\{iu : interactionuse$
$\bullet\ iu.interaction\} \bullet i.actor\} \bullet ac.actorBinding\} \bullet ab.instanceBinding\}$
$\cup \bigcup\{i : \bigcup\{ab : \bigcup\{ac : \{i : \bigcup\{iu : interactionuse \bullet iu.interaction\}$
$\bullet\ i.actor\} \bullet ac.actorBinding\} \bullet ab.binding\} \bullet i.instance\}$ [I 2]
$interactioninstance \neq \varnothing \Rightarrow instancesBoundToActor \subseteq members$ [I 3]

Critically, a *Cooperation* requires at least two *Instances* as members (see [I 1]). If a *Cooperation* applies *Interactions* through the concept of an *InteractionUse* then the *Actors* within these *Protocols* should only refer to members (i.e. *Instances*) of this *Cooperation*. For the formal specification of this invariant, we introduced a semantic variable called *instancesBoundToActor* that comprises all *Instances* that are bound to *Actors* of the corresponding *Protocols* that are used by the *Cooperation* (i.e. referred by *InteractionUse*). Thus, the variable *instancesBoundToActor* (see [I 2]) unions the set of *Instances* bound either directly through the concept of *Instance* or through the concept of *Binding* that binds *Instances* to *DomainRoles*. Furthermore, if the *Cooperation* uses *Interactions* for the purpose of communication, the set of *instancesBoundToActor* is a subset of the set of *members* specified at design time ([I 3]). Thus, a *Cooperation* provides only between associated *members* means for interaction.

Binding. A *Binding* defines which *Instances* are bound to which kind of *DomainRole*. The corresponding class schema is given below. Like a *Cooperation*, it inherits from the *NamedElement*. It includes the variables *instance* (which must not be empty, expressed by $\mathbb{P}_1$), *roleBinding* as well as *min* and *max* both of the type Integer. Critically, for all *instances* that are bound to a particular *DomainRole* (i.e. *roleBinding*) it is necessary that the *Agent* referred by the *instance* (i.e. *agentType*) performs this particular *DomainRole* the *instances* are now bound to (see [I 1]). Furthermore, we restrict the number of *Instances* that are bound to a *DomainRole* through the *min* and *max* variables. For instance, if *max* is greater than 0, invariant [I 3] states that the number of *Instances* is smaller or equal to *max* and greater or equal to *min*. However, if *max* is equal to 0, the number of *Instances* is an arbitrary number that is not fixed at design time.

Binding

NamedElement

$$instance : \mathbb{P}_1\, Instance,\ roleBinding : DomainRole\circledS,\ min, max : \mathbb{N}$$

$$\begin{array}{lr}
\forall\, i : instance \bullet roleBinding \in i.agentType.performedRole & [\mathsf{I}\ 1] \\
max \geq min & [\mathsf{I}\ 2] \\
max > 0 \Rightarrow max \geq \#instance \geq min & [\mathsf{I}\ 3]
\end{array}$$

InteractionUse. The class schema of *InteractionUse* is given below. It inherits from the class schema of *NamedElement* and includes two variables (i.e. *interaction* and *actorBinding*). Furthermore, we ensure with invariant [I 1] that all *ActorBindings* the *InteractionUse* refers to are unique, meaning that any two instances of them must be different with respect to their *names*, otherwise they are considered as being equal.

InteractionUse

NamedElement

$$interaction :\downarrow Interaction,\ actorBinding : \mathbb{P}_1\, ActorBinding\textcircled{c}$$

$$\begin{array}{lr}
\forall\, ab_1, ab_2 : actorBinding \bullet ab_1.name = ab_2.name \Rightarrow ab_1 = ab_2 & [\mathsf{I}\ 1]
\end{array}$$

ActorBinding. The concept *ActorBinding* binds *Instances* to *Actor* either directly or through a *Binding*. The class schema of *ActorBinding* is given below. It inherits from the class schema of *NamedElement* and its declarative part includes five variables (i.e. *instanceBinding*, *binding*, *actor*, *min*, and *max*). The variables *min*, and *max* again allow to restrict the number of *Instances* that are bound. Furthermore, we introduce a semantic variable called *instancesBound* that unions all *Instances* that are bound to the particular *Actor* (see [I 1]). Additionally, if *max* is greater than 0 invariant [I 3] states that the number of *instancesBound* is smaller or equal to *max* and greater or equal to *min*. However, if *max* is equal to 0 the number of *instancesBound* is an arbitrary number that is not fixed at design time.

ActorBinding

NamedElement

$instanceBinding : \mathbb{P}\, Instance \circledS, binding : \mathbb{P}\, Binding \circledS, actor : Actor$
$min, max : \mathbb{N}$
Δ [Semantic Variables]
$instancesBound : \mathbb{P}\, Instance$

$instancesBound = instanceBinding \cup \bigcup \{ b : binding \bullet b.instance \}$ [I 1]
$max \geq min$ [I 2]
$max > 0 \Rightarrow max \geq \# instancesBound \geq min$ [I 3]

Organization. An *Organization* is a special kind of *Cooperation* that also has the same characteristics as an *Agent*. Therefore, the *Organization* can perform *DomainRoles* and have *Capabilities*. As an *Organization* additionally inherits from *Cooperation*, it also has its own internal *Protocol*.

The class schema of an *Organization* is depicted below. It inherits from *Agent* and *Cooperation*. The declarative part does not consist of any variable.

However, the following invariants are specified: Firstly, the set of *requiredRoles*, and *bindings* ([I 1]) should be equal and greater or equal to two. Additionally, all *Instances* that are member of an *Organization* should perform a *DomainRole* that is required by the *Organization*.

Organization

Agent, *Cooperation*

$\# requiredRole = \# binding \geq 2$ [I 1]
$\forall\, m : members \bullet m.agentType.performedRole \cap requiredRole \neq \varnothing$ [I 2]

3.4 Role View

The *role view* (depicted in Fig. 5) covers the abstract representations of functional positions of autonomous entities within social relationships like for instance *Organizations* or *Cooperations*. The metamodel of the role view includes the concepts *Role*, *DomainRole*, and *Actor*, where *DomainRole* and *Actor* both inherit from *Role*. Furthermore, the role metamodel includes the concepts *Resource* (from the environment view) as well as *Capability* (from the agent view).

The class schema of *Actor* is depicted below. It inherits from the class schema of *Role* and includes three variables, i.e. *subactor*, *superactor*, and *actorBinding*, where *subactors* presents the different subsets of *Actors*, and *superactor* denotes for each subactor its corresponding parent actor. Furthermore, the variable *actorBinding* defines which kind of *DomainRoles* and *Instances* are bound to an *Actor* (see Fig. 4).

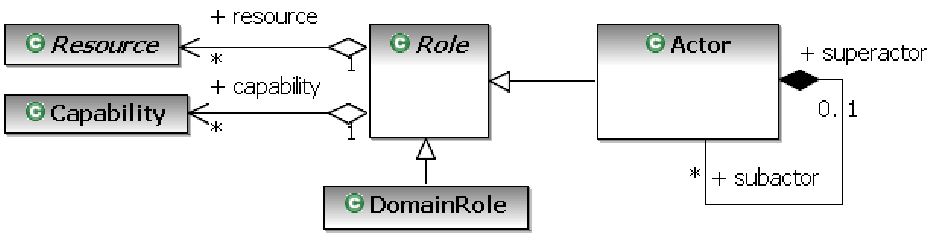


Fig. 5. The metamodel of the role view

Critically, each *Actor* should refer to one super actor at maximum (see [I 1]). As the *subactor* reference should be considered as a kind of specialization—meaning that the instances bound to a subactor are also part of the super actor—each *Actor* should have either no *subactor* or more than one *subactors* (see [I 2]). Furthermore, if an *Actor* has *subactors*, these *subactors* refer again to *Actor* as *superactor* (see [I 3]).

Actor

Role

$subactor : \mathbb{P}\, Actor©, superactor : \mathbb{P}_1\, Actor, actorBinding : \mathbb{P}\, ActorBinding$

$\#superactor \leq 1$ [I 1]

$\#subactor = 0 \lor \#subactor \geq 2$ [I 2]

$subactor \neq \varnothing \Rightarrow \forall\, a : subactor \bullet a.superactor = self$ [I 3]

3.5 Behavior View

The behavior view (depicted in Fig. 6) describes how simple actions are combined to more complex control structures or plans to achieve goals. The partial behavioral metamodel (see Fig. 6) includes the concepts *Behavior*, *Plan*, *Activity*, *Flow*, *InformationFlow*, *ControlFlow*, *StructuredActivity*, *Task*, *Sequence*, *Split*, *Loop*, *Parallel*, *Decision*, *ParallelLoop*, as well as *Send*, *Receive*, *Begin*, *End*, and *InternalTask*. Furthermore, the *Send* and *Receive* refer to a *Message* (from the interaction aspect).

A *Plan* specifies the *Agents'* internal processes. In general, *Plans* are executed by *Agents* in order to achieve their goals. However, the concept of a goal is not explicitly represented in the core of Dsml4mas. However, we intend to extend this core by further aspects also including an aspect for goal modeling. A *Plan* refers to a set of *Flows* that are contained in the process description and contains a set of *Activities* that are linked to each other via a *Flow*. Each *Flow* refers to a source *Activity* (which defines the start of the *Flow*) and a sink *Activity* (which defines the end of this *Flow*).

The class schema of a *Plan* is given below. It inherits from the class schemas of *Behavior* and *Activity*. The declarative part of the class schema consists of

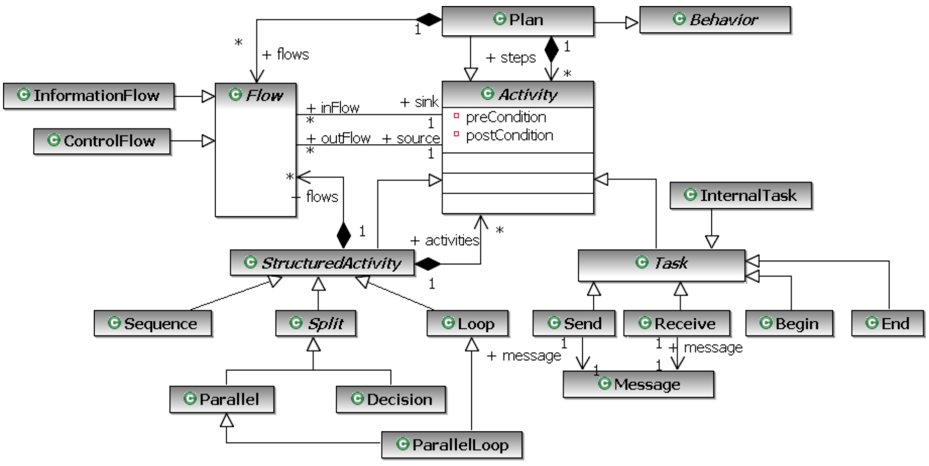


Fig. 6. The partial metamodel of the behavioral view

the variables *flows* and *steps*, where *flows* represents the set of *Flows* that are connecting *Activities*, and *steps* defines all associated basic or more complex *Activities*.

Furthermore, the declarative part presented in this schema consists of six invariants: Firstly, a *Plan* has exactly one starting *Activity* and several end *Activities* meaning that the set of *Activities* has exactly one *Activity* that has no *inFlow* ([I 1]) and several *Activities* that have no *outFlow* ([I 2]). However, the set of end *Activities* is not empty.

The operational semantics of a *Plan* are: When a *Plan* is entered, it becomes *active* meaning that the operation *enter* changes the variable *active* from false to true. When a *Plan* is exited, it becomes inactive meaning that the operation *exit* changes the variable *active* from true to false. Initially, the variable *active* is set to false (see operation *INIT*). Furthermore, a *Plan* is *entry* if it is *active* as well as the *preCondition* evaluates to true ([I 3]) and a *Plan* is *completed* if it is *active*, the ending *Activities* (i.e. *lastActivity*) of the contained *Activities* (i.e. *steps*) are completed, and the *postCondition* evaluates to true. Finally, we define three operations *ExecuteEntry*, *InnerExit*, and *Exit* which are used by the *ControlFlow* transitions when they enter or exit an *Activity*. Since executing a *Plan* should start with executing the start *Activity*, the operation *ExecuteEntry* invokes the operation *enter* of the first activity within the *Plan*. This guarantees that the containing *Activities* of a *Plan* are always *active* when the *Plan* is *active*. The *Exit* operation simply exits the *Plan* by invoking the operations *InnerExit* and *exit*.

Finally, we define the operational sequence in terms of invariants using the timed trace notation (see [9]). The invariants are described in the following notation: $\langle \neg\, var\rangle;\ \langle var\rangle \subseteq \langle \neg\, op\rangle;\ (\langle op\rangle \cup \langle op\rangle;\ \langle true\rangle)$ meaning that the operation *op* occurs immediately when the variable *var* evaluates to true. In the context of a *Plan*, the invariant defined in [I 5] states the operation sequence when a *Plan* is entered. The invariant ensures that the operation *ExecuteEntry* occurs immediately when the *Plan* is *active* and has been entered (i.e. *entry* evaluates to true). In this case the first activity of *steps* is executed. The invariant defined in [I 6] states the operation sequence when a *Plan* is exited. The invariant ensures that a *Plan* is only exited if the work on it has been completed. If this is the case, the operation *Exit* is immediately invoked.

Plan

$Behavior, Activity$

$flows : \mathbb{P} \downarrow Flow©, steps : \mathbb{P} \downarrow Activities©$

Δ [Semantic variables]

$firstActivity : \downarrow Activity, lastActivity : \mathbb{P}_1 \downarrow Activity, active : \mathbb{B}$

$entry : \mathbb{B}, completed : \mathbb{B}$

$firstActivity = head\{a : Activity \mid a.inFlow = \varnothing\}$ [I 1]

$lastActivity = \{a : Activity \mid a.outFlow = \varnothing\}$ [I 2]

$entry \Leftrightarrow active \land preCondition$ [I 3]

$completed \Leftrightarrow active \land postCondition \land \bigwedge s : lastActivity \bullet s.completed$ [I 4]

INIT

$\neg\, active$

enter

$\Delta(active)$

$\neg\, active \land active'$

exit

$\Delta(active)$

$active \land \neg\, active'$

$ExecuteEntry \mathrel{\widehat=} firstActivity.enter$

$InnerExit \mathrel{\widehat=} \bigwedge s : lastActivity \bullet s.completed \bullet s.exit$

$Exit \mathrel{\widehat=} InnerExit \fatsemi exit$

$\langle active \land \neg\, entry\rangle;\ \langle active \land entry\rangle \subseteq$
$\langle \neg\, ExecuteEntry\rangle;\ (\langle ExecuteEntry\rangle \cup \langle ExecuteEntry\rangle;\ \langle true\rangle)$ [I 5]

$\langle entry \land \neg\, completed\rangle;\ \langle entry \land completed\rangle \subseteq$
$\langle \neg\, Exit\rangle;\ (\langle Exit\rangle \cup \langle Exit\rangle;\ \langle true\rangle)$ [I 6]

4 Transferring Object-Z to OCL

In the last section, we presented a formal semantics specification of DSML4MAS. Existing Object-Z and Z tools can be used for checking, validating, and verifying the generated designs. However, this formal specification is also intended to support the system designer already during the design phase. Therefore, the formal specification should be closely linked to the tools used for designing MASs in accordance to DSML4MAS. To design in a graphical manner, we developed a graphical editor for DSML4MAS that bases on the abstract syntax defined by the PIM4Agents metamodel. For detailed information on the graphical editor, we refer to [11]. For the purpose of validating the design specified with the graphical editor, we integrated a checker that evaluates the models with respect to the semantics specification made with Object-Z. Using DSML4MAS correctly is important to ensure that the design can be automatically transformed to the underlying agent platforms on the PSM level. This checker is developed by manually transforming the Object-Z specification into a corresponding OCL (Object Constraint Language[2]) specification. This transformation is mainly done in an one-to-one manner and bases on [12] which presents a mapping from OCL to Object-Z. By performing the required reasoning with OCL inside the graphical editor, the system designer can easily model in accordance to the abstract syntax and semantics of DSML4MAS.

5 Related Work

There is huge literature on how to overcome the lack of semantics in modeling languages like UML, however, it is still a matter of ongoing research. For instance, the authors of [13] give a recent overview of existing approaches and evaluate them using previously developed criteria. One of the most prominent approaches is the UML 2 Semantics Project [14] that develops mathematically formalized semantics for UML. However, the lack of tool support for verifying and validating UML models has not been solved yet.

The most prominent approach for defining a formal semantics of an agent-based systems was proposed in [15]. The authors base their formal framework on Z (see [16] for an introduction to Z). The formal framework is composed of concepts and their schemas. However, state spaces and operations of agents are separated. In our approach, properties and operations of an entity are specified in the same Object-Z class. Object-Z as an object-oriented extensions for Z fits well in the context of model-driven development, which was in our case the main criterion for adoption.

The authors of [17] proposed an approach in which Object-Z is extended for specifying MASs. In accordance to them, AgentZ extends Object-Z with new constructs to enhance structuring and to accommodate new agent-oriented entities such as agents, organizations, roles and environments. This approach goes in the same direction as our approach, however, only the static semantics have been covered, whereas our approach also includes the dynamic semantics of DSML4MAS. Furthermore, we demonstrated that it is not necessary to extend Object-Z for specifying MASs in a formal manner.

An other approach is specified in [18] combining Object-Z and statecharts to define MASs as the authors consider Object-Z too weak for specifying the

[2] www.omg.org/docs/ptc/03-10-14.pdf

complex features associated with MASs. However, it is unclear whether the existing Object-Z tools can be used for checking, validating, and verifying the generated models.

Other formal or at least semi-formal approaches exist, like for instance the $i*$ framework proposed in [19]. A mapping between $i*$ and Z is discussed in [20].

6 Conclusion

This paper discusses parts of the formal semantics of Dsml4mas using Object-Z. For this purpose, for each concept in the PIM4Agents metamodel—which is the base for our domain specific modeling language—we introduced an Object-Z class that includes the abstract syntax as well as the denotational (static) semantics. In particular, the denotational semantics are defined by introducing additional semantic variables and invariants.

The dynamic semantics is specified in a similar way. It is first defined in a denotational manner, meaning that semantic variables and invariants were defined. Basing on these additional variables, the operational semantics is defined by introducing operations and invariants that restrict the order in which operations are executed using the time refinement calculus.

The following two core advantages of our approach can be identified: The formal specification provides a rigorous foundation of Dsml4mas. Therefore, by using existing formal verification tools, it may be possible to find errors, prove particular properties, and improve the quality of the language. That is an important language feature to minimize or even exclude errors. Furthermore, the graphical editor allows to integrate the abstract syntax as well as the formal semantics. Thus, the generated design can be validated at design time to exclude errors. Therefore, the formal semantics help to ensure that the users understand and use Dsml4mas correctly in order to apply code generators to close the gap between design and implementation.

References

1. Bauer, B., Müller, J., Odell, J.: Agent UML: A formalism for specifying multiagent interaction. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 91–103. Springer, Heidelberg (2001)
2. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: From stakeholder intentions to software agent implementations. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 465–479. Springer, Heidelberg (2006)
3. Papasimeon, M., Heinze, C.: Extending the UML for designing JACK agents. In: Proceedings of the Australian Software Engineering Conference, ASWEC 2001 (2001)
4. Bordini, R.H., Dastani, M., Winikoff, M.: Current issues in multi-agent systems development. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 38–61. Springer, Heidelberg (2007)
5. Hahn, C.: A platform independent agent-based modeling language. In: Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 233–240 (2008)

6. Smith, G.: The Object-Z Specification Language. Advances in Formal Methods, vol. 1. Kluwer Academic Publishers, Dordrecht (2000)
7. Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice-Hall International, Englewood Cliffs (1996)
8. Spivey, J.M.: The Z Notation: A Reference Manual, 2nd edn. Prentice Hall International Series in Computer Science, Englewood Cliffs (1992)
9. Smith, G., Hayes, I.J.: Structuring real-time Object-Z specifications. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2000. LNCS, vol. 1945, pp. 97–115. Springer, Heidelberg (2000)
10. Hahn, C., Madrigal-Mora, C., Fischer, K.: A platform-independent metamodel for multiagent systems. International Journal of Autonomous Agents and Multi-Agent Systems (2008)
11. Warwas, S., Hahn, C.: The contrete syntax of the platform independent modeling language for multiagent systems. In: Proceedings of the Agent-based Technologies and applications for enterprise interOPerability (ATOP 2008). Workshop hold at the Seventh International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2008 (2008)
12. Roe, D., Broda, K., Russo, A.: Mapping UML models incorporating OCL constraints into Object-Z. Technical Report 2003/9, Imperial College, 180 Queen's Gate, London (2002)
13. O'Keefe, G.: Improving the definition of UML. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 42–56. Springer, Heidelberg (2006)
14. Broy, M., Crane, M.L., Dingel, J., Hartman, A., Rumpe, B., Selic, B.: 2^{nd} UML 2 semantics symposium: Formal semantics for UML. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 318–323. Springer, Heidelberg (2007)
15. d'Inverno, M., Luck, M.: Understanding agent systems. Springer, New York (2001)
16. Spivey, J.M.: The Z notation: a reference manual. Prentice-Hall, Inc., Upper Saddle River (1989)
17. Brandão, A., Alencar, P.S.C., de Lucena, C.J.P.: AgentZ: Extending Object-Z for multi-agent systems specification. In: Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M. (eds.) AOIS 2004. LNCS, vol. 3508, pp. 125–139. Springer, Heidelberg (2005)
18. Hilaire, V., Koukam, A., Gruer, P., Müller, J.P.: Formal specification and prototyping of MAS. In: Omicini, A., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2000. LNCS (LNAI), vol. 1972, pp. 114–127. Springer, Heidelberg (2000)
19. Yu, E.: Towards modelling and reasoning support for early-phase requirements engineering. In: 3rd IEEE Int. Symp. on Requirements Engineering, pp. 226–235 (1997)
20. Vilkomir, S., Ghose, A., Krishna, A.: Combining agent-oriented conceptual modelling with formal methods. In: Proceedings of the Australian Software Engineering Conference, pp. 147–155 (2004)

Evaluating an Agent-Oriented Approach for Change Propagation*

Khanh Hoa Dam and Michael Winikoff

RMIT University, Australia
kdam@cs.rmit.edu.au, michael.winikoff@rmit.edu.au

Abstract. A central problem in software maintenance is *change propagation*: given a set of primary changes that have been made to software, what additional *secondary* changes are needed? Although many approaches have been proposed, automated change propagation is still a significant technical challenge in software engineering. In this paper we report on an evaluation of an agent-based approach for change propagation that works by repairing violations of well-formedness consistency rules in a design model. The results have shown that given a reasonable amount of primary changes, the approach is able to assist the designer by recommending feasible secondary change options that match the designer's intentions.

1 Introduction

A large percentage of the cost of software can be attributed to its maintenance and evolution [1]. The essence of software maintenance is change: in order to adapt a system to desired requirements (be they new, modified, or an environmental change), the designer makes changes to the system. In practice, those changes form a sequence of actions (addition, removal and modification) that contains some primary changes followed by additional, secondary, changes. Primary changes are usually identified based on the characteristics of the change requests and the designer's knowledge and expertise. After that, the designer ensures that other entities in the software system are updated to be consistent with these primary changes. As a result, secondary changes are then determined and performed, mostly by identifying and fixing inconsistencies in the design previously modified by primary changes. This process is known as *change propagation* [2] and is complicated, labour-intensive and expensive, especially in complex software systems that consist of many artefacts and dependencies [3].

Therefore, it would be desirable to have a tool that automates change propagation. However, we do not believe that a tool can fully automate change propagation because a tool cannot make decisions involving trade-offs and design styles where human intervention is required. However, a tool *can* be an assistant that helps the designer by providing feasible change propagation options.

* This work has been funded by the Australian Research Council under grant LP0453486, in collaboration with Agent Oriented Software. We would also like to thank Lin Padgham, Sebastian Sardina and other members of the RMIT Agent Group for discussion relating to this work.

Although a substantial amount of work has looked at the issue of change propagation, most of it has focused on source code (e.g. [3,2]). Recently, as the importance of models in the software development process has been better recognised, more work has aimed at dealing with changes at the *model* level (e.g. [4,5,6]) However, most existing work either fails to advocate effective automation or fails to explicitly reflect the cascading nature of change propagation, where each change (primary or secondary) can require further changes to be made.

We have developed an agent-based framework to deal with change propagation by fixing inconsistencies in a design. In other words, we identify change propagation options by finding places in the design where desired consistency constraints are violated by primary changes, and we then fix them. This approach represents options for repairing inconsistencies ("repair plans") using event-triggered plans, as is done in Belief-Desire-Intention (BDI) architectures.

The main focus of this paper is on an evaluation of the effectiveness of this approach in assisting with change propagation.

In the sections ahead, we first review this approach to change propagation including repair plan generation and execution (section 2). We then describe the Change Propagation Assistant (CPA) tool, which we have implemented and integrated with an existing modeling tool (section 3). Section 4 is the main focus of this paper in which an evaluation of the framework and our prototype tool is reported. We then discuss some related work in section 5 before concluding and outlining some future work (section 6).

2 An Overview of the Approach

Our approach to change propagation is to define consistency conditions using a (UML) **meta-model** and (OCL [7]) **consistency constraints**, and then use a library of **repair plans** to fix inconsistencies in the **design model**. Consistency constraints define conditions that all models must satisfy for them to be considered valid. These conditions may include syntactic well-formedness, coherence between different diagrams, and even best practices. Figure 1 depicts a very small excerpt of a UML metamodel [8] and below is an example consistency condition (in both OCL and logic) between class diagrams and sequence diagrams.

Constraint 1 *The name of a message (in sequence diagrams) must match an operation in a receiver's class (in class diagrams).*
Context Message inv c(self)*:*
self.receiver.base.operation→exists(op : Operation | op.name = self.name)
$\exists\, op \in self.receiver.base.operation : op.name = self.name$

There are two important properties of change propagation: (a) it is cascading, i.e. performing an action to fix an inconsistency can cause further inconsistencies which require further actions and (b) multiple choices, i.e. there are usually many ways of making the design consistent again. Those two properties are interestingly similar to the characteristics of the well-known and studied Belief-Desire-Intention architecture [9], where software agents have a library of plans ("recipes") which are triggered by events. Each plan specifies which event it is triggered by, under what conditions it should be

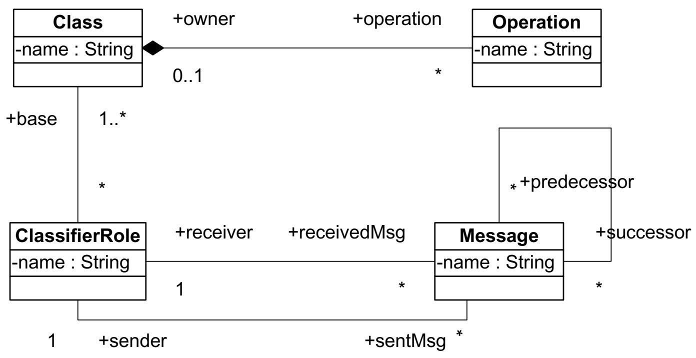


Fig. 1. A small excerpt of the UML metamodel

considered to be applicable (the "context condition"), and a plan "body": a sequence of steps that are what the plan does when it is executed. Steps in the plan body can create events, which result in further plans being triggered. A given event can have multiple plans that are triggered by it, and the plans' context conditions are used to select a plan to execute.

Based on that observation, repairing a violated constraint (an inconsistency) is represented as an event and the way to fix the violated constraint as (repair) plans. In previous work [10], the syntax for repair plans is defined based on AgentSpeak(L) [11]. Each repair plan, P, is of the form $E : C \leftarrow B$ where E is the triggering event; C is an optional "context condition" (Boolean formula) that specifies when the plan should be applicable[1]; and B is the plan body, which can contain sequences ($B_1;\ B_2$) and events which will trigger further plans (written as $!E$). We extend AgentSpeak(L) by allowing the plan body to contain primitive actions such as adding and deleting entities and relationships, and changing properties; and also to contain conditionals and loops. Our extensions are conservative, and the formal semantics for our repair plan language can be easily obtained by extending AgentSpeak's semantics [12], or by using the semantics of other related notations (such as CAN [13]).

Below is an example of repair plans for fixing constraint 1, i.e. $c(self)$ defined above. We use $c_t(self)$ to denote the event of making $c(self)$ true (similarly for $c1$). We also define the following abbreviations: $SE \overset{\text{def}}{=} self.receiver.base.operation$ and $c1(self, op) \overset{\text{def}}{=} op.name = self.name$.

P1 $c_t(self) : op \in SE \leftarrow !c1_t(self, op)$
P2 $c_t(self) : op \in Set(Operation) \wedge op \notin SE \leftarrow$ add op to SE ; $!c1_t(self, op)$
P3 $c_t(self) \leftarrow$ create $op : Operation$; add op to SE ; $!c1_t(self, op)$
P4 $c1_t(self, op) \leftarrow$ change $op.name$ to $self.name$
P5 $c1_t(self, op) \leftarrow$ change $self.name$ to $op.name$

[1] In fact when there are multiple solutions to the context condition, each solution generates a new plan instance. For example, if the context condition is $x \in \{1, 2\}$ then there will be two plan instances.

In the above example, there are three plans that make $c(\mathit{self})$ true (P1, P2, and P3). Plan P1, for instance, posts an event $c1_t(\mathit{self}, \mathit{op})$ which in turn can trigger either plan P4 and P5. The context condition of plan P1, $\mathit{op} \in \mathit{SE}$, indicates that at run time it can generate several plan instances, each for an operation belonging to the message's receiver's class. Informally, plan P1 fixes $c(\mathit{self})$ by either changing the name of an existing operation contained in the message's receiver's class to the name of the message (plan P4) or vice versa (plan P5).

In this change propagation framework, the repair plans are generated automatically (at design time) from the constraints and metamodel [10], and form a plan library which is used at run time. One key consequence of generating plans from constraints, rather than writing them manually, is that, by careful definition of the plan generation scheme, it is possible to guarantee that the plans generated are correct, complete, and minimal [10].

At run time, after primary changes are made to the design model, all of the constraints are evaluated, and violated constraints are repaired. Each violated constraint will usually have several repair plan instances for fixing it. While one repair plan may also fix other violated constraints, another may break constraints. As a result, selecting a repair plan needs to take into account its effect on other constraints. We have therefore defined an algorithm [14] which calculates a cost for each repair plan instance, taking into account its consequences, i.e. which constraints it may break or fix. The designer is presented with a set of the cheapest repair options, and they select one to apply to the design.

3 Implementation

We have recently implemented the Change Propagation Assistant (CPA), a prototype tool that demonstrates how the above approach works in practice. We have developed a PDT Interface Communicator component which provides an API that is used to integrate the change propagation tool with the Prometheus Design Tool (PDT[2]), a freely-available tool supporting designers using the *Prometheus* methodology [15] for building agent-based systems. The CPA uses the Dresden OCL Toolkit[3] to parse OCL constraints.

Our tool includes a Plan Creator component which generates (at design time) a repair plan library from the constraints and metamodel.

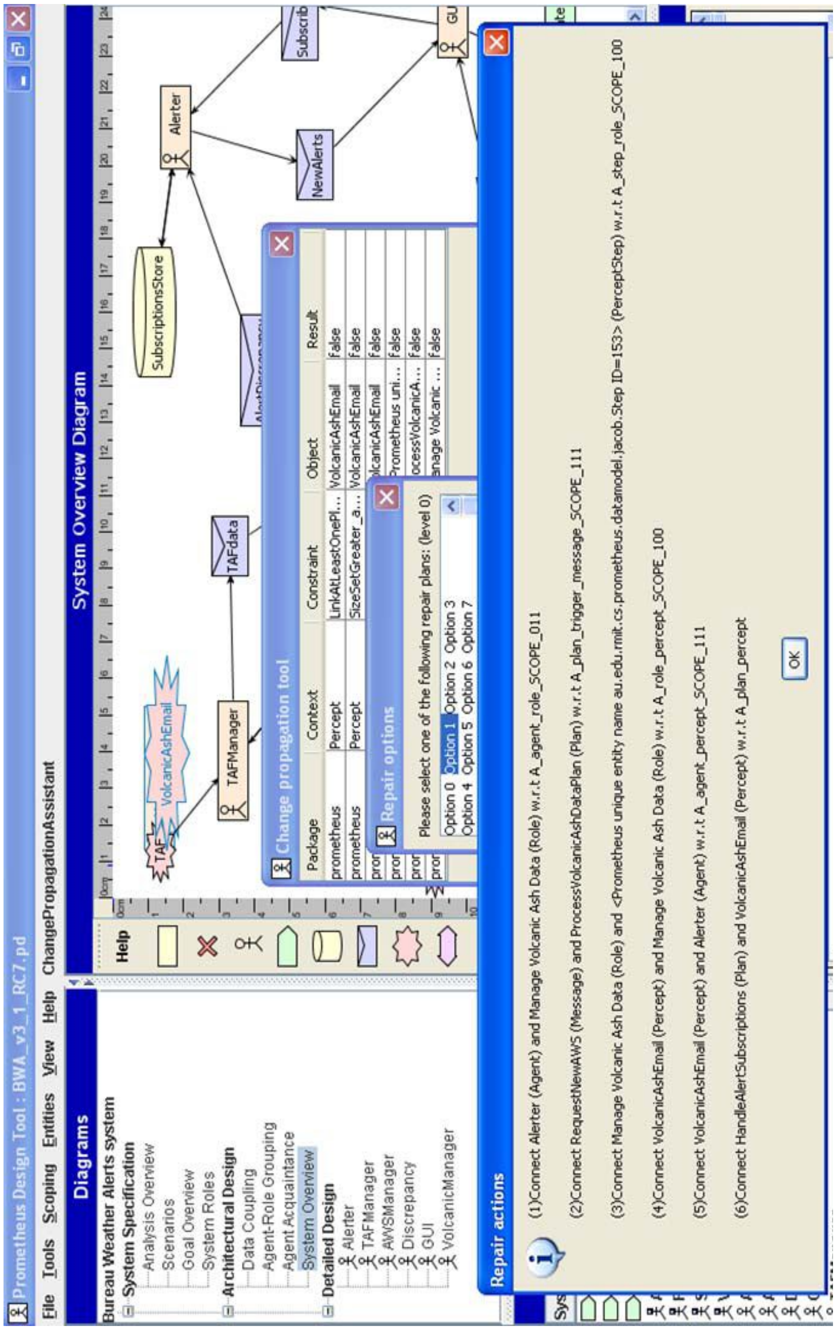
When the designer requests the CPA for help the PDT design model is converted by a model transformer component to a MOF[4]-compliant model which is stored in a NetBean MDR[5] repository. The Constraint Checker component identifies which constraints are violated and then instructs the Constraint Repairer component to find plans for fixing them, using the repair plans library. The Constraint Repairer performs the

[2] `http://www.cs.rmit.edu.au/agents/pdt`

[3] An open source project providing various tools for OCL `http://dresden-ocl.sourceforge.net`

[4] Meta Object Facility(MOF) is an OMG standard [16] for defining metamodels and metadata repositories.

[5] `http://mdr.netbeans.org/`



Change propagation tool

Package	Context	Constraint	Object	Result
prometheus	Percept	LinkAtLeastOnePl...	VolcanicAshEmail	false
prometheus	Percept	SizeSetGreater_a...	VolcanicAshEmail	false
pror			olcanicAshEmail	false
pror			Prometheus uni...	false
pror			ocessVolcanicA...	false
pror			anage Volcanic ...	false

Repair options

Please select one of the following repair plans: (level 0)

Option 0 Option 1 Option 2 Option 3
Option 4 Option 5 Option 6 Option 7

Repair actions

(1)Connect Alerter (Agent) and Manage Volcanic Ash Data (Role) w.r.t A_agent_role_SCOPE_011

(2)Connect RequestNewAWS (Message) and ProcessVolcanicAshDataPlan (Plan) w.r.t A_plan_trigger_message_SCOPE_111

(3)Connect Manage Volcanic Ash Data (Role) and <Prometheus unique entity name au.edu.rmit.cs.prometheus.datamodel.jacob.Step ID=153> (PerceptStep) w.r.t A_step_role_SCOPE_100

(4)Connect VolcanicAshEmail (Percept) and Manage Volcanic Ash Data (Role) w.r.t A_role_percept_SCOPE_100

(5)Connect VolcanicAshEmail (Percept) and Alerter (Agent) w.r.t A_agent_percept_SCOPE_111

(6)Connect HandleAlertSubscriptions (Plan) and VolcanicAshEmail (Percept) w.r.t A_plan_percept

OK

Fig. 2. Change Propagation Assistant Integrated with PDT

cost calculation and returns to the designer a set of cheapest repair options. If the designer accepts one of the options proposed then the Constraint Repairer instructs the PDT Interface Communicator to apply those changes to the current PDT design model.

Figure 2 shows our change propagation assistant tool integrated with the Prometheus Design Tool (PDT). When the CPA is invoked, it shows a list of violated constraints. The designer decides to repair these constraints and the tool comes back with a list of change options. The designer is able to view a sequence of change actions in each option and decides which option (if any) to use.

4 Evaluation

Having implemented the above approach for a change propagation assistant we would now like to perform an empirical evaluation of the effectiveness of the approach and tool. The key question is how well this approach works in practice and, specifically, how useful is it likely to be to a practising software designer who is maintaining and evolving a system?

Unfortunately, an evaluation to answer this question raises a number of challenges and questions such as: which methodology should be used? which application(s) should be used? what changes to the system should be done? and, how do we select primary changes to perform?

Our original plan was to use the UML design models, but the effort involved in implementing all of the constraints in the UML standard was beyond our resources, and so instead we have chosen to use the *Prometheus* [15] methodology for the design of agent systems. The Prometheus notation is simpler than UML, and in addition to local expertise, we had easy access to the source code of the Prometheus Design Tool (PDT), allowing the Change Propagation Assistant to be integrated with PDT. We used the Prometheus metamodel described by [17], and defined consistency constraints by examining the well-formedness conditions of Prometheus models, the coherence requirements between Prometheus diagrams, and best practices proposed by [15].

Our choice of application was the Bureau of Meteorology's multi-agent system for weather alerting (MAS-WA) [18]. This application was chosen because the prototype system developed by the Australian Bureau of Meteorology had been extended in a number of ways, and these extensions gave us well-motivated and realistic change scenarios to evaluate.

The purpose of the MAS-WA application is to monitor a range of meteorological data, and alert forecasters to situations such as extreme weather, inconsistencies between data sources, or changes to observed weather that contradict previously issued forecasts. We used a version of the system that simplified the application while retaining its key characteristics [19]. The simplified system monitored data from forecasts for airport areas (TAF) and from automated weather stations (AWS). TAF and AWS readings contain information about temperature, wind speed and pressure. The system issues alerts if there are significant differences between a prediction (TAF) and the actual weather (AWS). Figure 3 shows the system overview diagram for the application, as well as agent overview diagrams for the *Discrepancy* and *GUI* agent types.

Ideally, the evaluation would be done by giving the CPA to a group of selected users, who would be asked to work with the tool to implement requirement changes. However, due to time and resource limits, we were not able to conduct such a user study evaluation. In order to overcome this obstacle, we approached the evaluation by defining an abstract user behaviour in maintaining/evolving an existing design. We then simulated a real user by following this behaviour: repeatedly making changes to the design, and invoking and assessing the responses from the CPA tool.

Our model of abstract user behaviour below is based on the model of change propagation process of Hassan and Holt [20]. In this model, the developer is guided by a change request to perform primary changes (i.e. determine initial entity to change and change entity) and some partial secondary changes (i.e. determine other entities to change). When the developer cannot locate other entities to change, she/he consults a Guru (which could be a person, a tool or a suite of test; in our case, it is the CPA tool[6]), and if the Guru indicates that an entity was missed, then it is changed and the change propagation process is repeated for that entity. This continues until all appropriate entities have been changed.

In order to evaluate how useful the CPA is, we consider a given change to the system's design that was done in order to meet a new requirement. We view such a change (denoted by D, and not to be confused with the repair plans) as being a sequence of actions, with each action being a primitive change to the model such as adding or removing a link between entities. We then ask what proportion of the actions in the change was done by the user, and what proportion was done by the CPA.

However, this metric is not that simple to use, because it depends on the choice of change for a given requirement, and on the choice of primary change, i.e. how much of the change is done before invoking the tool. For each given change, D, we need to consider a range of possible primary changes P (with the actions in P being a subset of those in D). Now in fact, D is a sequence, and the abstract user behaviour below uses D sequentially, thus the possible primary changes are the initial segments of D.

We now define a process, based on the change propagation process of Hassan and Holt [20], that captures (abstractly) the designer's behaviour. In the following process D denotes the designer's planned change: a sequence of actions that transforms the existing design model so that it meets the new requirement. The designer performs an initial segment P of D (step 2), updates D by removing the performed actions (step 3) and then invokes the tool, which returns a set $\mathcal{O}$ of repair options (step 4). Each repair option C_i is a sequence of actions. At this point (step 5) the user may select one of the C_i (step 6) and apply it to the model (step 7), or he/she may decide that none of the C_i is suitable. Deciding whether a C_i is suitable is done by comparing it with the designer's plan: C_i is *compatible* with D if all actions in C_i are in D (formally[7] $\forall\, a \in C_i : a \in D$). If all the changes in D have been performed the process ends, otherwise the user continues to perform more primary changes (step 9). We use $D := D - P$ to denote removing the actions in P from the sequence D, and we use $P \sqsubseteq D$ to denote

[6] Unlike the Guru in their model, the CPA suggests not only entities to be changed, but also the specific changes to be made to them.

[7] Alternatively, if viewed as sets, $C_i \subseteq D$.

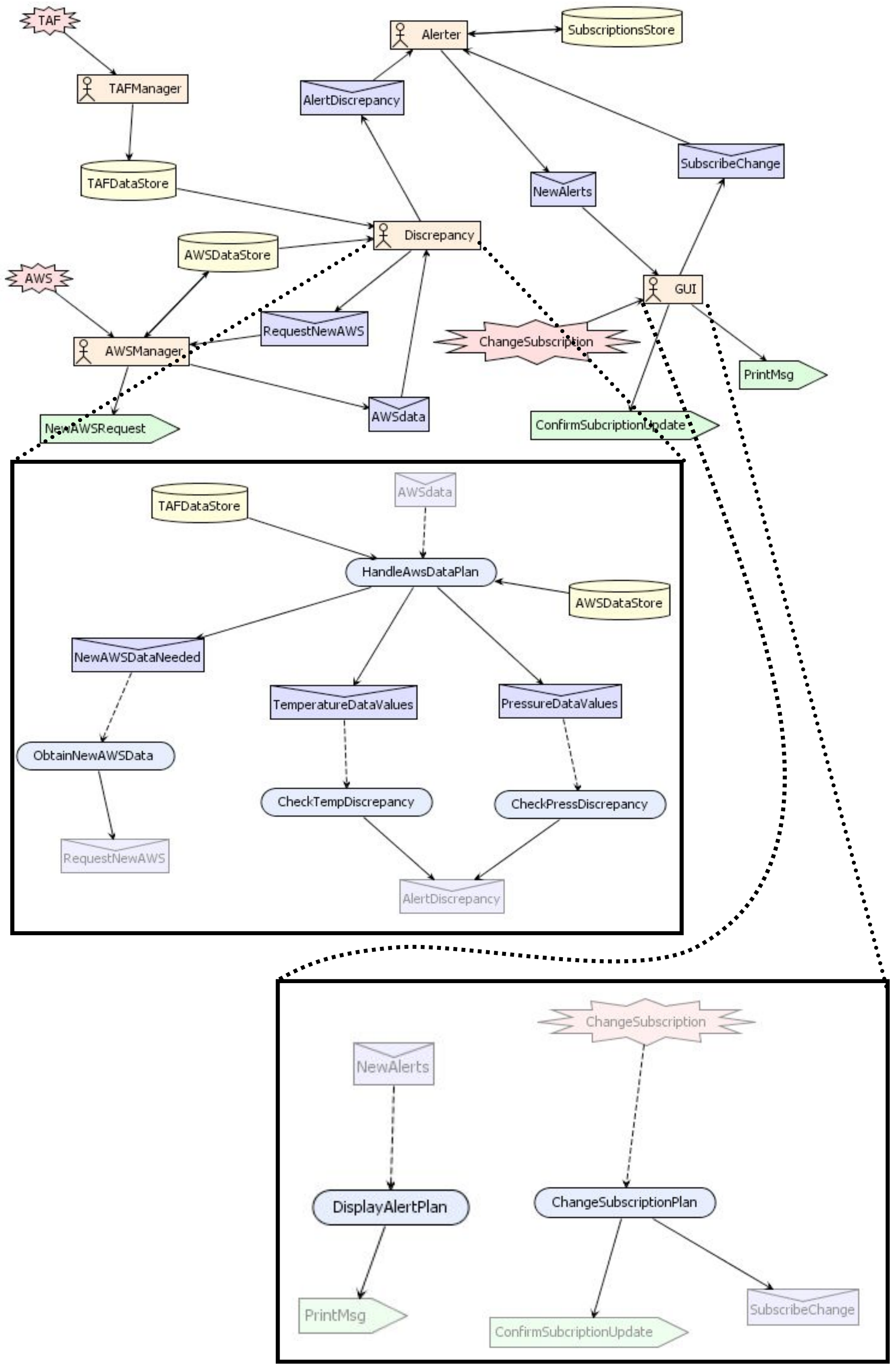


Fig. 3. MAS-WA Design

that P is an initial segment of D (formally $\exists\, X : P + X = D$, where $+$ is sequence concatenation).

```
1. given a planned change $D$ (sequence of actions)
2. select $P \sqsubseteq D$
3. do the actions in $P$ and update $D$ ($D := D - P$)
4. invoke the tool yielding $\mathcal{O} = \{C_1, \ldots, C_n\}$
5. if $\exists\, C_i \in \mathcal{O}$ where $C_i$ is compatible with $D$ then
6.     select a compatible $C_i \in \mathcal{O}$ (if more than one)
7.     do actions in $C_i$ and update $D$ ($D := D - C_i$)
8. end if
9. goto step 2 if $D$ is not empty.
```

We thus have the following evaluation process: for each new requirement we develop (and justify!) a change plan D, and then apply the process above, considering a partial change P that expands by one step at a time. When the process terminates we count how many actions ended up being in C_is along the way, compared with the total number of actions in D so we calculate the metric $M = | C \mid / \mid D |$ (where C is the union of the C_is).

Note that the value of M depends on our choices for P. Clearly, if P is the whole of D then there is nothing left for the tool to do, and M will be 0. In some of the cases below we will see that there is a "tipping point": until enough of D is done the tool cannot help, but once enough is done, the tool performs the remaining steps in D.

In addition to measuring M, an important factor in the usefulness of the tool concerns the repair options, $\mathcal{O}$. Specifically, we are interested in how many of the C_i in $\mathcal{O}$ are compatible with D, and in the size of $\mathcal{O}$ (since it is clearly better if the designer is not being asked to select an option from a very large list). We thus, in addition to M, also measure the number of options and how many of the options are compatible with D.

Finally, we need to select values for the basic costs of action. In this evaluation we do not explore a range of costs, but instead select what we believe are reasonable values: addition, creation and modification are assigned the same cost (e.g. 1) and we consider that deletion is not normally a desirable action, and so give it a higher cost (e.g. 5).

4.1 Results and Analysis

Change requests can be classified into several categories, depending on the dimensions we are looking at. Swanson initially identified three categories of maintenance: corrective, adaptive, and perfective [21]. These have since been extended with perfective maintenance[8] and has become an ISO/IEC standard [22].

We also view maintenance in terms of the type of modifications made to the software system. More specifically, they can be: (a) adding a new functionality/feature; (b)

[8] modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.

removing an existing functionality/feature; and (c) modifying an existing functionality/feature.

We introduce four requirement changes to the MAS-WA application that cover most of the change types, according to the above classification of changes. They include: logging all alerts sent to the forecast personnel (preventative and functionality modification); adding wind speed alerting (adaptive and functionality addition); implementing a variable threshold alerting (perfective and functionality addition); and adding volcanic ash alert (perfective and functionality addition). Due to space limitations, we describe only one change in detail, although we do present results for all of the changes.

Change: Implementing a Variable Threshold Alerting. Currently, the alerting levels are fixed and hard-coded. However, the forecast personnel wants to be able to adjust the alerting levels, e.g. different regions will show alerts based on different discrepancies. Hence, a new requirement that the forecast personnel should be able to set a threshold for alerting is requested.

To meet this change request, the "GUI" and "Discrepancy" agents must be changed. More specifically, the user's request for changing the thresholds will be represented by two percepts, each for temperature and pressure. A new plan is likely required to handle those percepts. A new data store that keeps the threshold information is introduced. Finally, the "Discrepancy" agent's plans for detecting discrepancies, i.e. "CheckTempDiscrepancy" and "CheckPressDiscrepancy", will need to use the new threshold data store. The designer's planned change D thus consists of the sequence[9]:

1. Create "SetNewTempThreshold" percept
2. Create "SetNewPressThreshold" percept
3. Create "ChangeThreshold" plan in "GUI" agent
4. Make "SetNewTempThreshold" percept to be a trigger of "ChangeThreshold".
5. Make "SetNewPressThreshold" percept to be a trigger of "ChangeThreshold".
6. Create a new "AlertingLevels" data
7. Link "GUI" agent to "AlertingLevels" data
8. Link "ChangeThreshold" plan to "AlertingLevels" data
9. Link "AlertingLevels" data to "Discrepancy" agent
10. Link "AlertingLevels" data to "CheckTempDiscrepancy" plan
11. Link "AlertingLevels" data to "CheckPressDiscrepancy" plan

Since the first three steps involves the creation of new entities and the later steps include actions related to those new entities, the tool does not return any compatible options until step 3 is performed (and the designer may well defer invoking the tool until s/he has created all three entities). One of the options proposed by the tool is making the two new percepts to be a trigger of the new "ChangeThreshold" plan. The tool does not suggest any further actions because the design is then consistent. The user then performs step 6 and the tool then recommends either steps 7 and 8 or steps 9 and 10 or steps 9 and 11. Assume[10] that the user chooses the option containing steps 7 and 8,

[9] Some variations in order are possible, but they do not affect the evaluation outcome

[10] If the user makes a different choice the overall M is still the same, just different actions are done by the tool.

Change	1	2	3	4	5	6	7	8	9	10	11	12	13	M
Logging	24_1	T	T											67%
Wind Speed Alert	50_0	6_1	T	T	U									40%
Variable Threshold	4_0	16_0	12_1	T	T	26_3	T	T	11_2	T	U			45%
Volcanic Ash	24_0	24_0	1_1	52_1	T	T	T	T	78_0	28_1	T	T	T	54%

Fig. 4. Evaluation Results

and then performs step 9. The CPA recommends either step 10 or 11. The user chooses one of these and has to manually perform the other.

Figure 4 shows the results of evaluation for the four changes. Each change has a row, where the entries marked with numbers show the situation for the *nth* step of the user's plan (D). An entry of the form n_m indicates that the tool returned n options (i.e. $\mathcal{O} = \{C_1 \ldots C_n\}$), where m of the C_i were compatible with D. An entry "T" indicates that this step is done by the tool, that is, it is part of a selected repair plan from an earlier step. An entry "U" indicates that the user performs this change; this occurs in two places where D is non-empty, but the design is consistent, and in this situation the tool cannot assist the user. The final column gives the value of the metric $M = \mid C \mid / \mid D \mid$. Overall, for the four changes the average value of M is approximately 50%, that is, compared with maintenance without our tool, the user would have to perform roughly twice as many change actions.

5 Related Work

There has been a lot of interest in addressing the issue of assisting software engineers to deal with software changes. In particular, change impact analysis has been extensively investigated but most of the work has focused on source code. Many of the impact analysis approaches are discussed in [3] and are typically used to assess the extent of the change, i.e. the artefacts, components, or modules that will be impacted by the change, and consequently how costly the change will be. Although these approaches are very powerful, they do not readily apply to design models [5]. In addition, our work focuses more on *implementing* changes, i.e. propagating changes between design artefacts in order to maintain consistency as the software evolves.

There have been several works that specifically target fixing inconsistencies in design models. The work by [6] provides a framework which automatically derives a set of repair actions from the constraint by analyzing consistency rules expressed in first-order logic and models expressed in xlinkit [23]. However, their work considers only a single inconsistency and consequently does not explicitly address dependencies among inconsistencies and potential consequences of repairing them, e.g. fixing one constraint can repair or violate others.

Recently, Egyed proposed an approach based on fixing inconsistencies in UML models [5]. The approach uses model profiling to locate possible starting points for fixing an inconsistency in a UML model. He also tried to use model profiling to predict the side-effects of fixing an inconsistency. His work, however, treats a constraint as a black box whilst we analyse the constraints to generate repair plans. Similarly the work of Briand et al. also looks at how to identify impacted entities during change propagation using UML models [4]. It defines specific change propagation rules (also expressed in OCL) for a taxonomy of changes. However, the major difference between both of these works and ours is that their approaches do not provide options to repair inconsistencies, but only suggest starting points (entities in the model) for fixing the inconsistency.

6 Conclusions and Future Work

We have presented a novel agent-oriented approach to deal with change propagation by fixing inconsistencies in the design models. The approach has been implemented in a form of a prototype tool (i.e. the CPA tool) that assists the designer in propagating changes. We have also evaluated the effectiveness of the approach by applying it to Prometheus using the design of a real application, MAS-WA.

The evaluation demonstrated that the approach is effective given that a reasonable amount of primary changes are provided. However, there are several threats to the validity of our study. For instance, although the set of changes are motivated by real change requests, and cover most of the change types, they may not be representative of all changes. We also need to test the approach with different application types. Additionally, there is scope for evaluation with other methodologies and notations (e.g. UML), with a range of basic costs, and, of course, with human subjects.

One issue that arises in the proposed approach relates to the use of inconsistency as a driver for change. As seen in the evaluation, not all changes result in inconsistency, and in these cases the approach will not be able to completely identify the desired secondary changes. An opposite issue is that, as argued by [24], not all inconsistency should be fixed; this is easy to deal with by simply allowing certain constraint types or instances to be marked as "to be ignored".

In some cases there may be a large number of repair options returned by the tool, which makes it hard for the user to select which one to use. In practice this can be dealt with by ignoring the tool's list of options and performing further changes (which often provides the tool with information that enables it to return fewer options). A better approach which needs to be investigated is reducing the number of options by "staging" questions. Suppose we need to link a percept with a plan and with an agent, then instead of presenting a set of options, where each option specifies both a plan and an agent (which gives a cross product), specify first the choice of agent, and then based on that choice ask for a choice of (relevant) plan.

Overall, our conclusion is positive since the evaluation shows that the approach is able to (on average) perform approximately half of the actions in maintenance plans, across a number of changes motivated by experience with a real application.

References

1. Vliet, H.V.: Software engineering: principles and practice, 2nd edn. John Wiley & Sons, Inc., Chichester (2001)
2. Rajlich, V.: A model for change propagation based on graph rewriting. In: Proceedings of the International Conference on Software Maintenance (ICSM), pp. 84–91. IEEE Computer Society, Los Alamitos (1997)
3. Arnold, R., Bohner, S.: Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos (1996)
4. Briand, L.C., Labiche, Y., O'Sullivan, L., Sowka, M.M.: Automated impact analysis of UML models. Journal of Systems and Software 79(3), 339–352 (2006)
5. Egyed, A.: Fixing inconsistencies in UML models. In: Proceedings of the 29th International Conference on Software Engineering (ICSE) (May 2007)
6. Nentwich, C., Emmerich, W., Finkelstein, A.: Consistency management with repair actions. In: ICSE 2003: Proceedings of the 25th International Conference on Software Engineering, pp. 455–464. IEEE Computer Society, Los Alamitos (2003)
7. Object Management Group: Object Constraint Language (OCL) 2.0 Specification (2006)
8. Object Management Group: Unified Modeling Langague Specification (UML 1.4.2, ISO/IEC 19501) (2005)
9. Rao, A.S., Georgeff, M.P.: An abstract architecture for rational agents. In: Rich, C., Swartout, W., Nebel, B. (eds.) Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning, San Mateo, CA, pp. 439–449. Morgan Kaufmann Publishers, San Francisco (1992)
10. Dam, K.H., Winikoff, M.: Generation of repair plans for change propagation. In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 132–146. Springer, Heidelberg (2008)
11. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
12. Moreira, A., Bordini, R.: An operational semantics for a BDI agent-oriented programming language. In: Meyer, J.J.C., Wooldridge, M.J. (eds.) Proceedings of the Workshop on Logics for Agent-Based Systems (LABS 2002), April 2002, pp. 45–59 (2002)
13. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative & procedural goals in intelligent agent systems. In: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR 2002), Toulouse, France, pp. 470–481 (2002)
14. Dam, K.H., Winikoff, M.: Cost-based BDI plan selection for change propagation. In: Autonomous Agents and Multi-Agent Systems (AAMAS) (2008) (to appear)
15. Padgham, L., Winikoff, M.: Developing intelligent agent systems: A practical guide. John Wiley & Sons, Chichester (2004)
16. Object Management Group: Meta Object Facility Specification, MOF 1.4 (2002)
17. Dam, K.H., Winikoff, M., Padgham, L.: An agent-oriented approach to change propagation in software evolution. In: Proceedings of the Australian Software Engineering Conference (ASWEC), pp. 309–318. IEEE Computer Society, Los Alamitos (2006)
18. Mathieson, I., Dance, S., Padgham, L., Gorman, M., Winikoff, M.: An open meteorological alerting system: Issues and solutions. In: Estivill-Castro, V. (ed.) Proceedings of the 27th Australasian Computer Science Conference, Dunedin, New Zealand, pp. 351–358 (2004)
19. Jayatilleke, G.B., Padgham, L., Winikoff, M.: A model driven development toolkit for domain experts to modify agent based systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 190–207. Springer, Heidelberg (2007)

20. Hassan, A.E., Holt, R.C.: Predicting change propagation in software systems. In: ICSM 2004: Proceedings of the 20th IEEE International Conference on Software Maintenance, Washington, DC, USA, pp. 284–293. IEEE Computer Society, Los Alamitos (2004)
21. Swanson, E.B.: The dimensions of maintenance. In: ICSE 1976: Proceedings of the 2nd international conference on Software engineering, pp. 492–497. IEEE Computer Society Press, Los Alamitos (1976)
22. ISO/IEC 14764: Information technology - software maintenance. ISO: Geneva, Switzerland (1999)
23. Nentwich, C., Capra, L., Emmerich, W., Finkelstein, A.: xlinkit: a consistency checking and smart link generation service. ACM Transactions on Internet Technology 2(2), 151–185 (2002)
24. Fickas, S., Feather, M., Kramer, J. (eds.): Proceedings of the Workshop on Living with Inconsistency, Boston, USA (1997)

Goal-Oriented Agent Testing Revisited

Erdem Eser Ekinci, Ali Murat Tiryaki, Övünç Çetin, and Oguz Dikenelli

Ege University, Department of Computer Engineering,
35100 Bornova, Izmir, Turkey
{erdemeserekinci,ovunc.cetin}@gmail.com
{ali.tiryaki,oguz.dikenelli}@ege.edu.tr

Abstract. Today multi-agent systems research is ready to be transferred to the industrial applications. But, testing is one of the most critical processes to increase the acceptability of such systems in industrial settings. In this paper, we introduce a goal-oriented testing approach based on test goal concept. This approach alleges that agent goals are smallest testable units in MAS's instead of agents unlike other agent testing approaches and tools proposed previously. Moreover, we introduce a testing tool, called as SEAUnit, that provides necessary infrastructure to support proposed approach.

1 Introduction

Multi-agent systems (MAS) are complex applications, which operate on open and distributed environments. So, developers need well established software processes for engineering industrial strenght MAS applications. As realized in traditional software development, testing is one of the most important processes in MAS development to produce industrial MAS applications. This fact has also been realized in MAS community and some testing processes [11,2] and supporting test tools [2,3] have been proposed in the literature.

To define a MAS testing process, one has to map the classical testing types (unit, integration, system and acceptance testing) to MAS development activities. Such a mapping forces us, first of all, to answer the most basic question; What is the unit on MAS development?

All of the previously proposed MAS testing processes and the tools describe agent as the smallest unit in MAS development and use event driven tool infrastructure(s) to verify the functional correctness of an agent. But an agent itself is a complex system which is too large to be considered as a unit. It may play many roles within the system including many goals and these goals are achieved through the complex infrastructure of agent that includes communication infrastructure, knowledge handling, planning etc. Hence, it is very difficult to localize errors and failures within such a large scale system by just observing the system events[8]. We need a smaller unit that simplifies the testing and error localization processes.

In this paper, we propose a new testing process, which takes the agent goal as the smallest unit to be tested. In general, AOSE methodologies identify system level goals in analysis phase, then those goals are partitioned and assigned to MAS architecture in the design phase. Agent executes these goals depending on their internal architecture (such as BDI) and collaboratively achieve the system level goals. So, agent level goal descriptions are the smallest methodological artifacts of any AOSE methodology.

After we identify agent goals as the smallest unit, we have to define how developers test these goals. we propose a new concept called as "test goal" for implementing unit tests. The test goals are explicit descriptions, which are written to ensure agent goals. This idea is inspired in the TDD practice of Extreme Programming (XP) [1] paradigm. Like in the XP, test goals are implemented with the supported agent programming language of the agent framework and stored as agent goals. Accordingly, coding tests as goals provide easily refactoring test code to source code and vice verse [15].

Test goals provide much more deeper testing opportunities than the event-based testing, since they use agent programming language to verify any agent's internal behavior. Moreover, requirements of agent collaboration can be verified within the test goal which makes the mock agent concept unnecessary. After the real agent is implemented, test goal is refactored to represent the new situation.

Based on the "test goal" concept and proposed process, a testing infrastructure has been implemented within the SEAGENT Multi-agent Development Framework[4]. This infrastructure lets developers to define test goals during goal implementation. So, it supports unit and integration testing during implementation cycle.

2 Background on Goal-Oriented Agent Testing

Automatic testing of agent systems depends on artifacts of the development methodology and used agent technology. So, to define a testing process one has to identify testable artifacts, that must be tested automatically, of agent development.

Nguyen and et al.[11] propose such a testing process by identifying different testing levels and activities involved within these levels based on TROPOS methodology [6]. There are three levels within the methodology, which are Unit Testing, Integration Testing and System Testing. In the unit testing level, they aim to test agents. Since the TROPOS methodology defines the agents own goals during development, they use the requirements of these goals for testing the agents. Goal dependencies of TROPOS goal model is used for the integration testing in a way that depended goals are allocated to different agents and collaboration of the agents are triggered in the scope of goals to test the integration.

Although Nguyen's approach is based on the goal concept, they aim to test the whole agent as the smallest testable unit. To test an agent, they send required test messages through a testing agent. But an agent has complex infrastructure. So it is not always possible to test the autonomy of agents and also internal behaviors of each individual goal from outside of the agent. The solution is to test the goals of agents internally.

In our testing process, the main artifacts that should be tested are system goals and agent goals. Agents play specified roles through executing their related goals. Moreover, agents collaborate through their roles (executing their goal in other words) to satisfy system level goals. One has to verify implementation of each agent goal firstly and then verify the system level goals by testing collaboration of the agents. As a conclusion, we take the agent goal as the smallest unit for goal-oriented agent testing process and define the test types within the process as follows:

– Agent Goal (Unit) Testing: Agent goal is the smallest testing unit and testing the agent goal verifies the valid execution of a goal in the single agent context. Agent context may include agent knowledge base, external environment that agent directly interacts. Agent's goal(s) should be tested by shielding the collaboration of other agent's goal(s) through support of testing environment which is discussed in Section 4.
– Integration Testing: Integration testing verifies functional acceptability of a system goal by focusing agent's goals collaboration. We have to test that system level goal requirements are satisfied by agent through their goal execution and collaboration.
– System Testing: System testing aims to validate of system execution on the real operational setting including the non-functional requirements such as performance, reliability, security etc. To perform system testing, one has to select a set of system goals that performs critical non-functional requirements.
– Acceptance Testing: Acceptance tests verify system functionality that is critical for the system users. So, the acceptance tests are applied to a set of system level goals that are selected by the system users.

After we setup the testing artifacts and type of the testing levels based on these artifacts, it is time to define when and how do we test these artifacts. The V-Model[1] is a well known model to map testing activities with development activities. According to the V-Model, artifacts of each development phase are tested by the related test types. A general goal-oriented testing process naturally executes acceptance test activities after the analysis phase of the development when system goals are identified. At this phase, test cases for selected system goals are defined and documented. System test cases requires a defined agent architecture. Hence, system test activities are performed after the design phase when agent architecture that satisfies the system goals are defined.

The main difference of our approach is to collaborate unit testing and integration testing activities with the implementation phase in a way that agent goals and system goals are implemented using the test goal concept. The details of how the test goals are defined and implemented is discussed in following sections.

[1] The Development Standards for IT Systems of the Federal Republic of Germany, www.v-modell-xt.de

3 Goal-Oriented Testing Approach

Goal-Oriented multi-agent system development methodologies and goal-based agent architectures aim to develop multi-agent system applications based on the goal concept. General view of such methodologies can be summarized like that requirements of an application are first transferred to system level goals, then system level goals are decomposed to agent level goals, which satisfy system level goal cooperatively. So, to ensure acceptability and validity of these goals, testing process of a MAS application should verify system and agent level goals.

In this paper we proposed a new approach for testing system and agent level goals. In this approach, our claim is that tests of such goals should be implemented as goals also. The goal that is used for testing is called as "Test Goal" for emphasizing difference between test and source goals. A test goal is separated conceptually to three main sub-goals, which are named as *setup, goal under test* (**GUT**) *and assertion* goals, as shown in Figure 1. The *setup* goal prepares pre-conditions and provisions, which make the execution of GUT possible. When the setup goal is satisfied than the GUT is executed. After the GUT execution, assertion goal is started and it checks expected post-conditions and outcomes of the GUT execution. Since assertion and setup goals are implemented as separate goals, they may have complex implementation, which may gather information from different sources (other agents, environment) and may assert other agents internal knowledge and states. According to complexity of the tests, these goals may be also decomposed to sub-goals and some source goals or other test goals can be re-used for the assertion implementation.

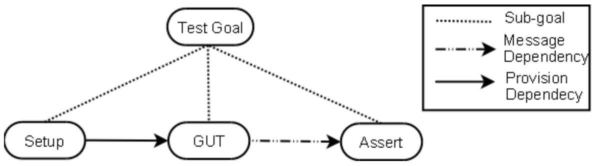


Fig. 1. Unit Test of a Goal

A test goal's level depends on the covered GUT. If the GUT is a type of an agent goal than it is verified in unit testing level. On the other hand, if GUT is a system goal, which is composed of agent goals, than it is tested in the integration testing level. Our test goal concept makes it possible to verify both unit and integration level testing using the same approach, which is implementing a test goal to satisfy a goal (either agent or system level goal).

On the other hand the system level testing is a more comprehensive testing phase and it has many types of testing such as validity, stress, security, regression vs. It is not possible to achieve all these types of tests by the test goal concept except validity testing. Which types of system testing techniques and acceptance testing can be covered by the test goal approach is not in the scope of this paper.

Following section defines how test goals can be implemented for unit and integration testing levels in an agent architecture, which uses HTN planning paradigm to implement agent goals. Although the test goal approach is exampled on HTN paradigm, this idea can be applied to any goal-oriented agent architecture.

4 Applying Goal-Oriented Testing on HTN-Based Agent Architecture

HTN(Hierarchical Task Network)[13] planning paradigm has been extensively used in well known agent frameworks such as RETSINA [12], DECAF [7] and SEAGENT[4]. The HTN has pre-defined plan structure that is modeled in the design time, and then these structures are executed by HTN planner[5] at run-time. The HTN's pre-defined structure includes *Behaviors*, *Action*s and their data and flow dependencies. *Behaviors* are complex HTN tasks, which define the composition of other behaviors and actions are executable and primitive tasks. At run-time the planner reducts behaviors to sub-tasks iteratively and executes the actions.

In agents that use the HTN paradigm, HTN plans satisfy agent goals. To test an HTN plan in our testing approach, we implement separate HTN plans for setup and assertion goals and these plans are linked dynamically at run-time for achieving test goal.

To explain details of our testing process, we use a simple example, which is shown in Figure 2. In this example a system goal is achieved by the collaboration of two agent goals. Each agent goal, which is implemented with an HTN plan, is executed in an agent in scopes of different roles. In the HTN structures shown in the figure, letter B indicates *behavior* and letter A indicates *actions.* As shown in the figure, the *Consumer Goal* is implemented by *B1* behavior in the scope of *R1* role. When the B1 is started, initial action *A1* of the *B1* triggers the *Provider Goal* by sending a message that contains a provision of *A3.* The *B2* finishes its job with executing A4 action and the A4 sends the result message to the *A2.* When the *A2* receives the message, it executes and finishes the *Consumer Goal.* This example is used to define how our goal-oriented testing approach is applied on the HTN implementation in the following sub-sections.

4.1 Unit Test Detail

As mentioned in previous section, in the goal-oriented testing approach we claim that agent goals should be considered as primitive modules of a MAS application and should be verified by unit tests. Based on the HTN structure, highest level behaviors fulfill the agents' goals. So, the highest level behaviors identify requirements of the test goal in a way that setup goal has to prepare pre-conditions of the GUT, similarly assert goal verifies post-conditions of the GUT based on the requirements of highest level behavior.

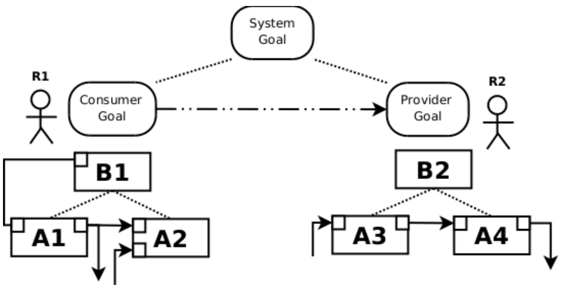


Fig. 2. HTN Sample Plan

Figure 3 illustrates that how the *Consumer Goal* of the *R1* role (comes from our original example shown in the Figure 2) is tested using our approach. As shown the figure, each sub-goal of the test goal (setup,GUT,assert goals) has separate HTN plans and these sub-goals have dependency links between each other. So, at run-time these goals are executed according to the dependencies in a way that each depended goals are linked dynamically during goal test execution.

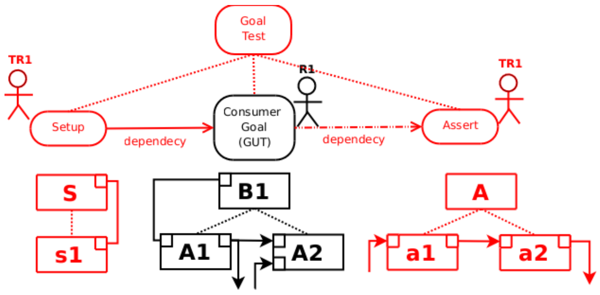


Fig. 3. Unit Test

Goal dependencies between *Setup, GUT* and *Assert* goals also require dependencies between their HTN plans. For example, setup plan *S* prepares provision of the B1 behavior of GUT. After *the Setup* goal execution, goal matching engine identifies the dependencies between *Setup* and *Consumer Goal (GUT)* and initiate the plan of *Consumer Goal* by providing the B1's provision with outcome of S. So, such a mechanism requires a goal matching engine, which executes sub-goals by following dependencies of a goal and separate planning engine that executes plans of related goals.

Similarly, the *Consumer Goal* and *Assert Goal* has a different kind of dependency (dotted link represents message dependency) and this dependency indicates that message of *Consumer Goal* (A1's message) must be link to the *Assert Goal* (a1 action of A behavior). When the *A1* task sends a message, the message is received by the same agent and then the goal matching engine of the agent identifies the *Assert Goal* that is capable of handling this message and initiate the plan of that goal. Then the *a1* task of *Assert Goal* verifies the functionality of *A1* task.

On the other hand, the *GUT* may require external provisions from cooperated goals (these goals may locate in the same agent or different agents). To verify the *GUT,* such provisions must be provided at run-time and in our example this case is shown as message dependency between *a2* and *A2* actions. It is important to note that this mechanism make the mock agent concept (this is the core concept of other automatic testing tools) unnecessary since required information about a *GUT* can be gathered by task(s) of an assert goal. Additionally mock agents requires message dependency between agents. If an agent goal does not send and receive external messages, it is not possible to verify such a goal with mock agents.

4.2 Integration Test Detail

We already define aim of integration testing level as verifying system goals. Since system goals include cooperation of agent goals of separate roles, we have to verify results of this cooperation. An integration test goal contains setup, system goal (as GUT) and assertion goals similar with unit testing. Although sub-goals of GUT (it is a system goal in this case) can be located in different agents, we don't aim to test messaging infrastructure of the agent platform. So, we execute the integration test goal within an agent to ensure functional correctness of the cooperation.

The main difference of an integration test goal from a unit test goal is that setup and assertion goals of integration test goal contains sub-goals. Naturally, setup goal of an integration test goal prepares pre-conditions of each sub-goal of the GUT. Similarly, sub-goals of the assert goal check the correctness of functionality and cooperation of GUT's sub-goals.

Figure 4 indicates an integration test goal structure, which verifies our base example shown in Figure 2. Since system goal (GUT) contains two sub-goals (*Consumer* and *Provider Goal*s), the *Setup* goal requires two related sub-goals (*Consumer* and *Provider Setup Goal*s) for preparing pre-conditions of each separate sub-goals of the GUT. The execution of an integration test goal is managed by goal matching engine as in the unit testing case. During the execution of the integration test goal, the matching engine resolves that *Setup* goal has two

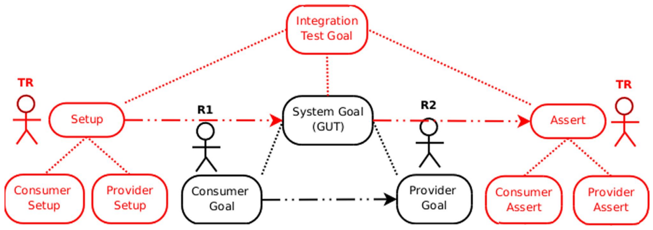


Fig. 4. Integration Tests

independent sub-goals and initiates their plans. After the plans of sub-goals are completed, the matching engine follows dependencies between *Setup* and *GUT* goals and then launches the *Consumer Goal* and its all provisions are provided. Dependencies between sub-goals of *GUT* and sub-goals of *Assert* are managed by the matching engine and their related plans are executed in a required order.

5 The Architecture of SEAUnit

The SEAUnit test tool helps to developers to run test goals of our proposed goal-oriented testing approach. The SEAUnit architecture has been engineered on the top of SEAGENT Multi-Agent Framework [4]. The test tool's execution mechanism is simple since it directly executes the test goals using SEAGENT's infrastructure. The SEAGENT has two critical model for implementing SEAUnit architecture, which are planning engine and goal matching engine. The planning engine is based on HTN paradigm [13] and execute HTN plans to satisfy agent goals. The details of the planning engine architecture can be found in [5]. The matching engine's main responsibility is to select and launch goals for planning to satisfy incoming requests. The SEAGENT matching engine can manage dependencies of sub-goals of an agent goal by cooperating with the planning engine.

The SEAUnit is implemented as Eclipse plug-in and incorporated in the SEAGENT development environment, which is also an Eclipse plug-in. The architecture of the tool and related modules of the SEAGENT framework are illustrated in Figure 5.

As seen in the figure, all responsibilities for testing are taken by two subpackages; *Test Runner* and *GUI.* To run a test goal, the *Test Runner* initializes an instance of agent within the SEAGENT platform. This agent becomes responsible to execute test goals. Then the *Test Runner* triggers the agent by sending an objective for satisfaction of the test goal.

During execution process of the test goal, the *GUI* listens events of the agent. These events are created by the *Event Manager* module of the SEAGENT Agent and are collected by the *GUI.* Two types of events of *Event Manager* have been used to monitor test goal execution; finished goal, and erroneous goal. Finished goal and erroneous goal events indicate that the goals under execution are satisfied or not. The *GUI* holds the knowledge of the test goal that are inprogress. According to the collected events about the test goal, such as finished, exceptions (about communication, knowledge base or other runtime exceptions) and removed from task queue, the *GUI* reports the results of the assert goals to the developer.

At first glance, this event listening mechanism may mislead the reader that the test tool uses events like other agent testing researches. But the important point that should be paid attention is that *GUI* listens events only about the test goals/tasks for reporting purposes.

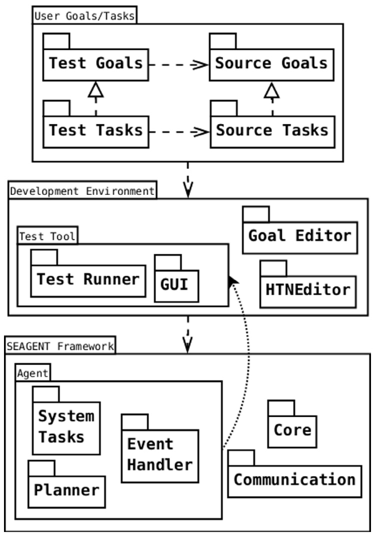


Fig. 5. Test Tool Infrastructure

6 Case Study

We have developed a case study to show how the proposed testing approach and the SEAUnit testing tool work on a real example. The case study is focused on a single system goal of a tourism information system called as *RoomReserve*. The model of ReserveRoom system goal, its sub-goals and corresponding HTN plans are represented in Figure 6.

During the design phase, it is decided that the *ReserveRoom* system goal includes agent goals; *RequestRoomReservation* of a *Customer* role and *RegisterRoomReservation* of a *Hotel* role. When agent of the *Customer* role decides to reserve a room according to the agent user's preferences, it takes *RequestRoomReservation* goal and its corresponding plan *BHReservationRequest* into the execution. The execution of the *BHReservationRequest* plan starts the cooperation with player agent of the *Hotel* role by sending request messages. To participate in the cooperation, the *Hotel* role takes *RegisterRoomReservation* goal and goal's corresponding plan *BHRegisterReservation* plan into execution.

To define execution of agent goals, details of the HTN plans, shortly mentioned above and represented in the Figure 6, must be explained. These HTN plans, *BHRegisterReservation* and *BHRegisterReservation*, are developed by the SEAGENT development environment's HTN Editor. These plans cooperate in the boundary of a simple request protocol. At run-time, the *BHReservationRequest* behavior takes room description as a provision and this provision is carried to the *ACRequestRegistration* action with an inheritance link. After, the ACRequestRegistration action has the responsibility of enveloping this room description to a FIPA message and sending it to the to the *Hotel* role. Next action

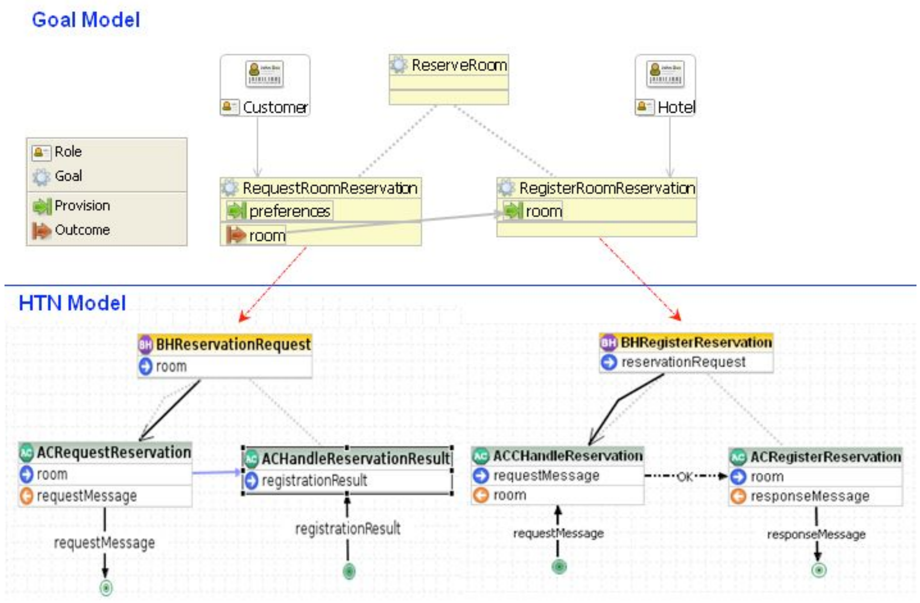


Fig. 6. Tourism Information System Goal Model

is called as *ACHandleReservationRequest*, which takes a response message from the *Hotel* role and displays to the user. When the request message, which contains the room definition, is arrived to the agent of Hotel role, the agent planner executes the *BHRegisterReservation* behaviour. This behaviour handles the request message with its *ACCheckReservation* action. This action picks out the room definition from the message and outcomes it to the *ACRegisterReservation* action. This action checks the availability of the room and books or rejects the request according to the availability of the room.

In the scope of the case study, unit testing process forthe *RegisterRoomReservation* agent goal is explained. We implement *TestRoomReservation* test goal to ensure validity of the *RequestRoomReservation* goal. Figure 7 illustrates goal model of the *TestRoomReservation* goal. As seen in the figure, there are two participant role; *Test* and *Hotel* roles. The *TestRoomReservation* goal comprised of three sub-goal *Setup*, *RegisterRoomReservation* and *CheckRoom* goals. The *CheckRoom* goal is also composed of sub-goals of *CheckRoomAvailability* and *Assert*. When the *TestRoomReservation* goal is triggered, firstly the *Setup* goal is executed and the *Setup* goal starts to cooperate with the *RegisterRoomReservation* goal of *Hotel* role. In the cooperation, the *Setup* goal of *Test* role requests to book a room and sends this request message to the *RegisterRoomReservation* of *Hotel* role. After the *Setup* and *RegisterRoomReservation* goals' cooperation finished, the *CheckRoom* goal is took into progress. In the scope of the *CheckRoom* goal, the *Assert* goal starts and cooperates to *CheckRoomAvailability* goal of the *Hotel* role to ensure the reservation result.

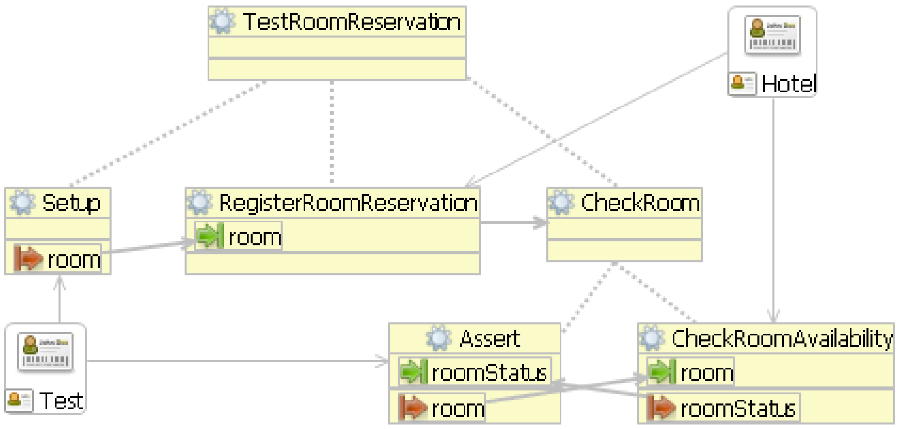


Fig. 7. Goal Model of Test Role

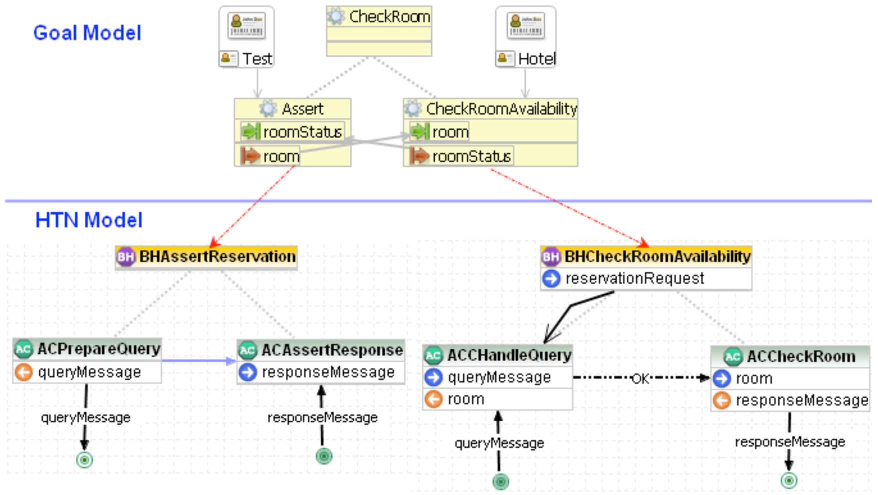
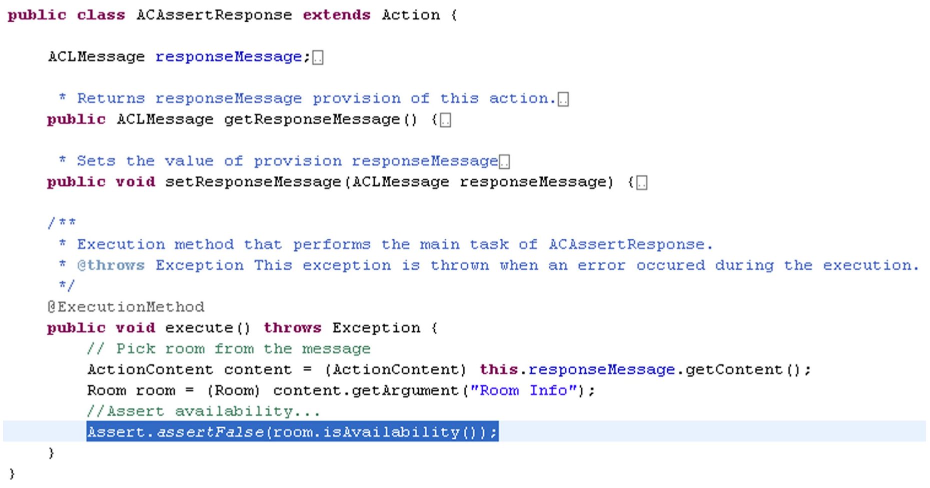


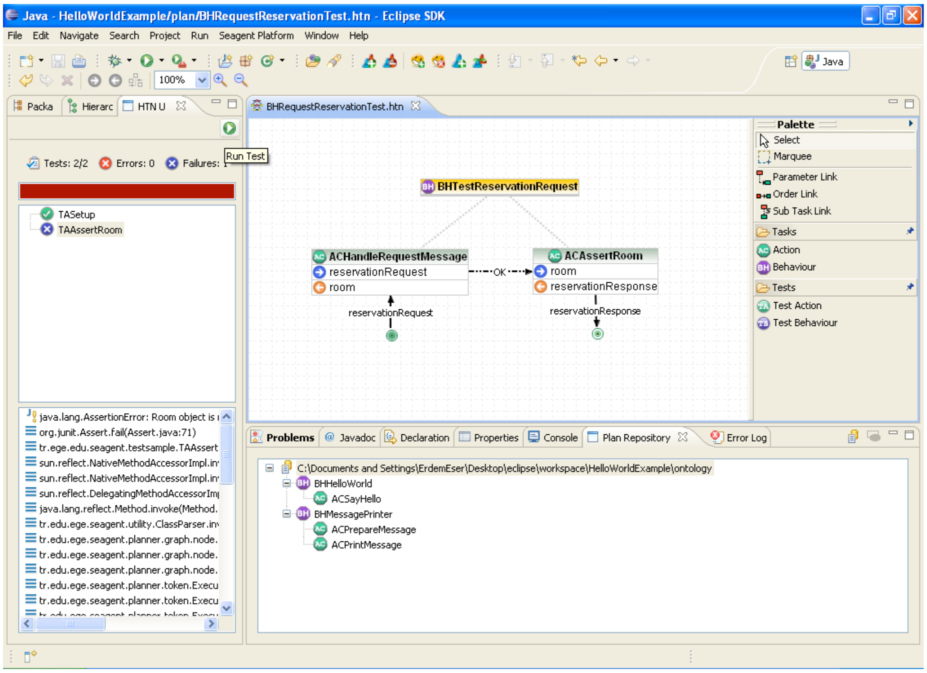
Fig. 8. CheckRoom Goal and Corresponding Plans

To implement the test goal of *TestRoomReservation* goal, one has to be define setup and assertion goals. In Figure 8, the assert goal of the test goal is shown but the *Setup* goal is skipped, since its plan is very simple that just prepares a room description as an input. The plan of the *Assert* goal is named as *BHAssertReservation,* which is composed of two sub-tasks, *ACPrepareQuery* and *ACAssertResponse*. The *ACPrepareQuery* action prepares query message to ask room availability to the agent that plays Hotel role. The *ACHandleQuery* action receives the query message, sent from the *ACPrepareQuery* action, and



```
public class ACAssertResponse extends Action {

    ACLMessage responseMessage;

     * Returns responseMessage provision of this action.
    public ACLMessage getResponseMessage() {

     * Sets the value of provision responseMessage
    public void setResponseMessage(ACLMessage responseMessage) {

    /**
     * Execution method that performs the main task of ACAssertResponse.
     * @throws Exception This exception is thrown when an error occured during the execution.
     */
    @ExecutionMethod
    public void execute() throws Exception {
        // Pick room from the message
        ActionContent content = (ActionContent) this.responseMessage.getContent();
        Room room = (Room) content.getArgument("Room Info");
        //Assert availability...
        Assert.assertFalse(room.isAvailability());
    }
}
```

Fig. 9. Assert action code



Tests: 2/2 Errors: 0 Failures: 1

TASetup
TAAssertRoom

```
java.lang.AssertionError: Room object is
org.junit.Assert.fail(Assert.java:71)
tr.ege.edu.seagent.testsample.TAAssert
sun.reflect.NativeMethodAccessorImpl.in
sun.reflect.NativeMethodAccessorImpl.in
sun.reflect.DelegatingMethodAccessorIm
java.lang.reflect.Method.invoke(Method.
tr.edu.ege.seagent.utility.ClassParser.in
tr.edu.ege.seagent.planner.graph.node.
tr.edu.ege.seagent.planner.graph.node.
tr.edu.ege.seagent.planner.graph.node.
tr.edu.ege.seagent.planner.token.Execu
tr.edu.ege.seagent.planner.token.Execu
```

C:\Documents and Settings\ErdemEser\Desktop\eclipse\workspace\HelloWorldExample\ontology
- BHHelloWorld
  - ACSayHello
- BHMessagePrinter
  - ACPrepareMessage
  - ACPrintMessage

Fig. 10. SEAUnit Tool View

picks out the room description from the message and passes it to the *ACCheck-Room* action. The *ACCheckRoom* queries its local knowledge according to the room definition and sends back the availability result to the *ACAssertResponse* action. Lastly, the *ACAssertResponse* verifies whole reservation operation is successful or not.

The *ACAssertResponse* action, represented in Figure 9, is the action that contains only assertion code. The figured action is implemented conveniently to the SEAGENT framework's action template without any extension about testing framework. After the provision of the action is supplied, its execute method is called by the planner. Similarly to the body of the execute method, the room object is picked out from the message, that is sent from the Hotel role, and asserted to verify room is not available. At run-time, if the assertion fails, an assertion exception is thrown and events about these type of exceptions are collected by the test tool GUI. This GUI is shown in Figure 10. As seen in the figure, the GUI shows the result of the assert goal in the right side of the view. As as last sentence of the case study, it can be said that this integrated testing environment makes test goal development very easy.

7 Evaluation and Conclusion

The desire of implementing an agent test tool raised while our group was developing MAS applications. During development efforts, verification of implemented goals became a problem within the group. This requirement forced us to develop a testing framework for agent-oriented development, since each developer tends to observe the correctness of his/her implemented goal.

Initially we implemented a testing framework [14] that was based on the event-based architecture similar with the other testing tools in literature. But a lot of difficulties were run across and some of them were really strict. First difficulty was defining new types of events when the SEAGENT infrastructure was modified. This problem is also observed in the literature [9]. This made testing test tool and agent infrastructure highly coupled. In the new testing architecture, the testing tool does not cope with the events of planning engine. Test goals are executed as ordinary plans by the planning engine. The *Event Manager* handles the all events of infrastructure and test tool just listens the *Event Manager* to report to test goals' status. This design decouples the test tool from planer's internal execution.

Another observed problem of the event-based testing architecture is inability of reusing previously implemented goals. In the new architecture, since test goals are implemented as an HTN plans, any source goals or test goals can be reused easily by linking during goal development.

As a conclusion, we change the architecture of our test tool to overcome mentioned problems of event-based architecture. The new architecture and testing approach makes the testing process easily applicable for any goal-oriented agent development methodologies. Since explicit test goals are implemented using same programming paradigm (HTN planning in our case), developers can easily adopt themselves to unit and integration test coding.

We also observed that our integrated testing environment(SEAUnit) facilitates test goal development and makes it possible to get rapid feed-backs about goal functionality. So, the integrated test tools supports the test-driven goal development style for developers. Also test goals can be used for documentation purposes which simplifies understanding and maintenance of agent goals [10].

References

1. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley Professional, Reading (2004)
2. Caire, G., Cossentino, M., Negri, A., Poggi, A., Turci, P.: Multi-agent systems implementation and testing. In: Fourth International Symposium: From Agent Theory to Agent Implementation, Vienna, Austria (EU), April 14-16 (2004)
3. Coelho, R., Kulesza, U., von Staa, A., Lucena, C.: Unit testing in multi-agent systems using mock agents and aspects. In: SELMAS 2006: Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems, pp. 83–90. ACM Press, New York (2006)
4. Dikenelli, O., Erdur, R.C., Gumus, O., Ekinci, E.E., Gurcan, O., Kardas, G., Seylan, I., Tiryaki, A.M.: Seagent: A platform for developing semantic web based multi agent systems. In: AAMAS, pp. 1271–1272. ACM, New York (2005)
5. Ekinci, E.E., Tiryaki, A.M., Gurcan, O., Dikenelli, O.: A planner infrastructure for semantic web enabled agents. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part I. LNCS, vol. 4805, pp. 95–104. Springer, Heidelberg (2007)
6. Giunchiglia, F., Mylopoulos, J., Perini, A.: The tropos software development methodology: processes, models and diagrams. In: AAMAS 2002: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, pp. 35–36. ACM, New York (2002)
7. Graham, J.R., Decker, K., Mersic, M.: Decaf - a flexible multi agent system architecture. Autonomous Agents and Multi-Agent Systems 7(1-2), 7–27 (2003)
8. Hanks, S., Pollack, M., Cohen, P.: Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. AI Magazine 14(4), 17–42 (1993); Also published as TR 93-06-05, University of Washington
9. Hanks, S., Pollack, M.E., Cohen, P.R.: Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. Technical Report TR-93-06-05 (1993)
10. Meszaros, G.: XUnit Test Patterns: Refactoring Test Code. Addison-Wesley, Reading (2007)
11. Nguyen, C.D., Perini, A., Tonella, P.: A goal-oriented software testing methodology. In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 58–72. Springer, Heidelberg (2008)
12. Sycara, K., Paolucci, M., Velsen, M.V., Giampapa, J.: The retsina mas infrastructure. Autonomous Agents and Multi-Agent Systems 7(1-2), 29–48 (2003)
13. Sycara, K., Williamson, M., Decker, K.: Unified information and control flow in hierarchical task networks. In: Working Notes of the AAAI 1996 workshop Theories of Action, Planning, and Control (August 1996)
14. Tiryaki, A.M., Öztuna, S., Dikenelli, O., Erdur, R.C.: Sunit: A unit testing framework for test driven development of multi-agent systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 156–173. Springer, Heidelberg (2007)
15. van Deursen, A., Moonen, L., van den Bergh, A., Kok, G.: Refactoring test code. In: Marchesi, M., Succi, G. (eds.) International Conference on Extreme Programming and Flexible Processes (2001)

Experimental Evaluation of Ontology-Based Test Generation for Multi-agent Systems

Cu D. Nguyen, Anna Perini, and Paolo Tonella

Fondazione Bruno Kessler
Via Sommarive, 18
38050 Trento, Italy
{cunduy,perini,tonella}@fbk.eu

Abstract. Software agents are a promising technology for today's complex, distributed systems. Methodologies and techniques that address testing and reliability of multi agent systems are increasingly demanded, in particular to support automated test case generation and execution.

A novel approach, based on agent interaction ontology, has been recently proposed and integrated into a testing framework, called *eCAT*, which can generate and evolve test cases automatically, and run them continuously.

In this paper, we focus on the experimental evaluation of an ontology-based test generation approach. We use two BDI agent applications as case studies to investigate the performance of the framework as well as its capability to reveal faults.

1 Introduction

Testing of Multi-Agent Systems (MAS) faces challenging issues due to basic features of these systems. MAS are distributed systems, they are autonomous and deliberative, they operate in an open world, which requires context awareness, and they can communicate using high level interaction protocols which may require facing semantic interoperability problems. All these features are known to be hard not only to design and to program [1], but also to test.

Recent studies on MAS testing [18,6] have proposed two testing frameworks, working on different agent platforms, both based on JUnit and sharing the idea of using mock agents that interact with the agents under test by sending messages to them, according to a given interaction protocol. The replies of the agents under test are then evaluated against expected behaviors. Although these approaches represent an important step toward the automation of MAS testing, they mainly address test case execution. Issues related to test cases generation, which concern how to effectively generate valid and invalid input data to thoroughly exercise the agents' behavior, are still largely unexplored in MAS testing.

Agent behaviors are often influenced by messages received. Hence, at the core of test case generation is the ability to build sequences of messages that exercise the agent under test so as to cover most of the possible running conditions. Often, failures appear only after long execution periods and under specific conditions and environmental contexts. A key feature of agent testing is the possibility to carry on long interactions with the agents under test. This demands for automated approaches to test case production, so as to overcome the necessarily limited number of cases considered in a manually defined test suite.

Automated test case generation requires the ability to fill-in message templates with input data that are both meaningful and diverse enough to stimulate the alternative reactions of the agent under test. Manual and purely random input generation are deemed to cover a limited portion of the input space for different reasons: manual input data are necessarily a small set; random input data are generated without taking the semantics of the messages into account. For example, a string can be easily generated randomly, by picking up a random sequence of characters, but such an automatically generated input is very unlikely to match, e.g., a valid airline name or destination, which might be necessary to construct a valid message processed by the agent under test.

We have recently addressed the problem of meaningful input value generation, taking advantage of agent interaction ontologies, which define the semantics of agent interactions [13]. The resulting ontology-based approach allows to automatically generate both valid and invalid test inputs, to provide guidance in the exploration of the input space, and to obtain a test oracle against which to validate the test outputs. We have integrated this techniques in our MAS testing framework, called $eCAT$, which supports continuous testing and automated test case generation [12], so that test cases are generated and executed continuously, resulting in a fully automated testing process, which can run unattended for long periods of time. On the one hand, continuous testing enables extensive exploration of the space of MAS behaviors, on the other hand, ontology-based test generation guides this exploration toward the most interesting regions of the space of input data that appear in meaningful messages.

In this paper, we focus on the experimental evaluation of the ontology-based test generation technique. Two different-size MAS applications have been chosen as case studies: a book-trading system and a system that supports bibliography research. We evaluate the performance of the framework as well as its capability to reveal faults.

The paper is structured as follows. Section 2 briefly discusses state-of the art research on MAS testing and contrast our approach with it. Section 3 recalls some background notions on our continuous testing framework and on agent interaction ontologies. Section 4 recalls the ontology-based test generation approach, which have been previously proposed in [13], and its implementation in the $eCAT$ framework. Results from the experimentation on the two case studies are discussed in Section 5, with the aim of evaluating the effectiveness of the proposed approach. Finally, Section 6 summarizes the main outcomes of this work.

2 Related Work

MAS verification, debugging and testing have been addressed in a few studies [2,18,6]. Focusing on MAS testing, Rouff [16] proposes a special tester agent for testing other agents individually or within the community they belong to. We share with Rouff [16] the choice of testing a MAS using an agent, and we go even further by separating the testing responsibility from the monitoring responsibility, the latter being assigned to monitoring agents. This makes our framework easily applicable to distributed MAS. In addition to that, in our framework the tester agent executes continuously, thanks to the possibility of running new test cases that are generated automatically.

Tiryaki et al. [18] propose a test-driven MAS development process that supports iterative and incremental MAS construction. A testing framework, built on top of JUnit and Seagent [7], is used to support the approach. The framework allows writing automatically executed test cases that exercise single agent behaviors as well as interactions among agents. Similarly, Coelho et al. [6] introduce an approach for MAS unit testing built on top of JUnit and JADE [17]. Both approaches involve mock agents, which simulate real agents, to interact with the agents under test. However, a number of mock agents needs to be implemented in order to test every role of agents under test, this makes these approaches expensive and scalability becomes a problem.

Zhang et al. [19] discuss unit testing for agent systems. Different from traditional software systems, units in agent systems are more complex in the way that they are triggered and executed, such as plans are triggered by events. A model-based testing approach has been introduced that use Prometheus [14] models to obtain information about faults. A testing framework that supports unit testing is also introduced.

A common approach in MAS debugging tools, consists of collecting information during the execution of MAS and visualizing them to support the identification of the errors and of their causes. The following approaches focus on debugging agent interactions. The ACLAnalyser [4] tool analyzes runs on the JADE [17] platform. It intercepts all messages exchanged among agents and stores them in a relational database. It exploits clustering techniques to build agent interaction graphs that support the detection of missed communication between agents that are expected to interact, unbalanced execution configurations, overhead data exchanged between agents. This tool has been enhanced with data mining techniques to process results of the execution of large scale MAS [3]. Lam and Barber [11] presents the Tracing Method and accompanying tool to help debug agents by explaining actual agent behavior. The Tracing Method captures dynamic run-time data, creates modeled interpretations in terms of agent concepts (e.g. beliefs, goals, and intentions), and analyzes those models to gain insight into both the design and the implemented agent behavior.

The main difference of our approach from existing works is that we aim at automating test case generation, also exploiting agent interaction ontologies for managing valid and invalid inputs and guiding exploration of the input space.

Moreover, test cases are executed continuously and in an unattended way, so that testing can last for a long time in the background.

3 Background

3.1 The *eCAT* Testing Framework

The *eCAT* testing framework for MAS, generates test cases and executes them continuously [12]. The architecture of the tool is depicted in Figure 1, with its two main components: the *Tester Agent* and the *Monitoring Agents*. Since agents communicate primarily through message passing, the *Tester Agent* can send messages to other agents to stimulate a behavior that can potentially lead to fault discovery. In turn, the *Monitoring Agents* observes the reactions to the messages sent by the *Tester Agent* and, in case these are not compliant with the expected behavior, e.g. conditions violated or crashes happen, it notifies the developer team of the revealed fault. Usage of an autonomous tester agent allows for an arbitrary extension of the testing time, that can proceed unattended and independently of any other human-intensive activity. This is a relevant function, since the behavior of a MAS can change over time, due to the mutual dependencies among agents and to their learning capabilities, and a single execution of a test case might be inadequate to reveal faults. Continuous testing of a MAS requires that the *Tester Agent* has the capability to evolve existing test cases and to generate new ones, with the aim of exercising and stressing the application as much as possible, the final goal being the possibility to reveal yet unknown faults.

Two automated test case generation techniques were initially provided with the *Tester Agent*, namely random generation and evolutionary mutation generation [12]. They have been recently enhanced with the ontology-based generation

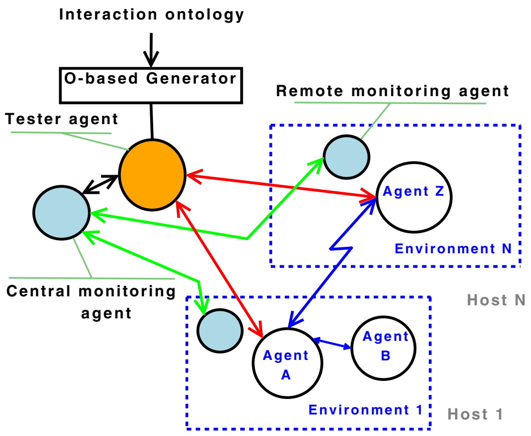


Fig. 1. *eCAT* architecture, including ontology-based input generator

technique recalled in Section 4 [13]. The *Tester Agent* invokes the generator (*O-based Generator*, see Figure 1) to continuously generate test cases and run them while the *Monitoring Agents* observe their behaviors and informs the *Tester Agent* of faults, if any occur. Moreover, the *Tester Agent* verifies whether the messages received from the agents under test conform to the ontology or not. In the latter case, the *Tester Agent* notifies the developer team of the revealed fault.

eCAT has been implemented as an Eclipse[1] plug-in. It supports testing agents implemented in JADE [17] and JADEX [15], and the input ontology formats are those supported by Protégé[2] like OWL.

3.2 Agent Interaction Ontology

In order for a pair of agents to understand each other, a basic requirement is that they speak the same language and talk about the same things. This is usually achieved by means of an ontology, namely an *interaction ontology*. Popular multi-agent platforms like JADE [17], JADEX [15], widely support the use of ontologies. They provide tools for generating code from ontology documents, thus, reducing the development effort, and for runtime binding of the message contents with concepts defined in an ontology.

A common structure of an interaction ontology involves two main concepts (also known as *Class*es): `Concept` and `AgentAction`. Sub-classes of `AgentAction` define actions that can be performed by some agents (e.g., `Propose`), while sub-classes of `Concept` define common concepts understandable by agents that interact (e.g., `Book`). These sub-classes can have multiple properties of different type. For each class, the user can define a number of individuals (or instances) of the class. For example, `Book` can have *title* as a property, and a particular book having the title "Testing Agents" may be an instance of `Book`.

A specific agent action can now be built, based on the shared understanding of the concept `Book`. For example, an agent *Buyer* could send an ACL message of type `REQUEST` to agent *Seller*, with the following content:

(Propose (Book :title "Testing Agents") :price 135.7)

The message is understood by both agents thanks to the shared interaction ontology.

4 Ontology-Based Test Generation

Let us consider a book-trading multi-agent system in which *Seller* and *Buyer* agents negotiate in order to sell and buy books. There could be multiple sellers and buyers that want to sell or buy the same book at the same time, so the goal of the sellers is to choose a buyer that proposes the highest price whereas the

[1] `http://www.eclipse.org`

[2] Available at `http://protege.stanford.edu`

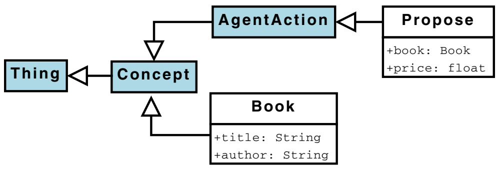


Fig. 2. The book-trading interaction ontology, specified as UML class diagram

goal of the buyers is to choose the seller with the cheapest price. Let us assume that these agents use the FIPA Contract Net protocol [9] and the interaction ontology presented in Figure 2. The ontology consists of a concept `Book` having two properties *title* and *author* and an agent action `Propose`, to proposes a *price* for a *book*.

Rules can be added to the ontology properties in order to restrict admitted values. For example, the *price* property in Figure 2 may be constrained to be within 0 and 2000. The related OWL rule is the following:

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#price"/>
  <owl:hasValue ...>min 0 and max 2000</owl:hasValue>
</owl:Restriction>
```

In the course of negotiation, a buyer initiates the interaction by sending a call for proposals for a given book (an instance of `Book`) to all the sellers that it knows of. Upon reception of a call, sellers check whether the book is available and make price proposals (instances of `Propose`). The initiating buyer, then, selects the seller with the minimum price and sends an accept for the proposal to the selected seller, or it changes the proposal and re-sends it to the seller, according to the FIPA Contract Net protocol.

Testing the *Buyer* and *Seller* agents accounts for generating instances of the messages that each agent is supposed to be able to process and let the *Tester Agent* send them to the proper agents. The *Tester Agent* continues the interaction in accordance with the selected protocol and generates new messages whenever needed. The *Monitoring Agents* observe and record any deviation from the expected behavior of the agents under test. Hence, the problem for the *Tester Agent* is how to generate meaningful messages in the course of the interaction. We take advantage of the interaction ontology for this purpose.

4.1 Domain Ontology and Ontology Alignment

By domain ontology we mean ontologies that exist in a specific domain of interest, not necessarily being interaction ontologies. For instance, there are ontologies that describe books and all related information such as title, author, category, year of publication and the like. A number of domain ontologies are available on the Internet (a useful ontology search service is available at: `http://swoogle.umbc.edu`).

Ontology alignment [8,10] refers to techniques aimed at finding relationships between elements of two ontologies. It can be used to map classes, properties, rules etc. of one ontology onto another one, and eventually to compare or transfer data from one to the other. Several tools are available that support ontology mapping (e.g., *Prompt*, available at `http://protege.stanford.edu/plugins/prompt/prompt.html`).

In order to generate meaningful testing data, i.e., data that represent real instances of ontology classes in the domain of interest, we use ontology alignment, so as to augment the agent interaction ontology with instances. This is achieved by mapping an existing domain ontology onto the MAS interaction ontology. Since domain ontology may come with a large amount of associated data, by ontology alignment we can augment the interaction ontology with a rich set of diverse data, that can be used for test case generation.

In the *book* domain, it is easy to find hundreds of instances of the class `Book` on the Internet that can be aligned with the interaction ontology thanks to tools such as *Prompt*. For more details on ontology mapping, the interested reader can refer to the paper by Euzenat et al. [8].

4.2 Ontology-Based Test Generator

The basic features of the ontology-based test cases generation technique, previously presented in [13] are described in the rest of this section.

Valid and invalid inputs. The task of the ontology-based test generator consists of completing the content part of the message the *Tester Agent* is going to send to the agent under test. For each concept to be instantiated in the message, the generator either picks up an existing or creates a new instance of the required concept. The test generator generates no input value if the interaction protocol prescribes that a value from a previously exchanged message must remain the same.

Then, the selected instance is encoded according to the proper content codec (for the message content) and is made available to the *Tester Agent*. As an example, the following excerpt shows an XML-encoded content of a message that contains information about a proposal for a book, including the `Propose` action:

```
<root ... xmlns="jadex.examples.booktrading.ontology"/>
   <Book n:id="2" title="Introduction to MultiAgent Systems"
                  author="Michael Wooldridge"/>
   <Propose n:id="1" price="47.50" r:book="2"/>
</root>
```

When new instances are generated, the test generator selects one from those available in the ontology based on the number of usages of each instance, so as to increase diversity and explore the input space more extensively.

In the case when no ontology instances are available, valid test inputs can be still generated taking into account information, such as rules and property datatypes, specified in the interaction ontology. For example, based on the rule about the *price*, the generator can generate any value in the range from 0 to 2000 as a valid input value to be processed by the *Seller* or *Buyer* agents.

More generally, for the properties of *Numeric* datatype, we can exploit the boundaries of the datatype, as well as the rules that limit the values of the specific property, to generate valid input values. For the properties of *string* datatype, we can only exploit the list of allowed values, if available. Most of the times, meaningful values for properties of string datatype are hardly generated without the help of an ontology.

Invalid input generation is based on a specific set of rules and datatypes that appear in the interaction ontology and complement those for the valid inputs. For instance, when boundaries are specified for numeric properties, the generator goes beyond them deliberately. According to the book-trading ontology described above, the test generator knows that the property *price* is of datatype *float* and that there is a rule stating that *price* must be between 0 and 2000. The generator may produce the invalid values -1, 2001 to test both sides of the boundaries. For string properties, the generator produces null (or empty) strings as potentially invalid values. Other options available to the generator are to randomly modify a valid input (taken from the available ontology instances) or to randomly generate a new one in order to try to produce an invalid value. The full list of valid input generation rules is provided in [13].

Message generation. When generating the full message, the test generator applies a set of input combination rules, such that, for valid messages, the only possibility is to use only valid input values, while for invalid messages, the generator can choose either to have only invalid values, or to have interleaved valid and invalid values, or to have just one invalid value. Rule selection follows the general criteria of maximizing diversity.

Input space exploration. The generator uses coverage information to decide how to explore the input space. The test generator gives priority to concepts and instances never selected before. When instances are reused, if possible the generator selects instances with low reuse frequency. When invalid inputs are produced, the generator chooses the so-far least-used invalid input generation rules. From the coverage perspective, the input space corresponding to each ontology concept is divided into valid and invalid regions. The generator gives priority to uncovered or least-used regions, thus avoiding test cases belonging to the same category and increasing the coverage rate.

Ontology as test oracle. The expected behavior of the agents under test is checked, not only, by a set of OCL constraints, but we can also enrich such constraints with a set of constraints automatically derived from the interaction ontology. In fact, the message content sent by the agents under test is expected to respect the rules and datatypes specified in the ontology for each concept instantiated in the message. Whenever the *Tester Agent* receives a message content that is invalid according to the interaction ontology, a fault is notified to the developer team. For example, when the *Tester Agent* sends a call for proposal for a book, the *Seller* agent must reply with a message whose content belongs to `Propose` and complies to its rules and datatypes. Otherwise, an error is detected.

5 Case Study

We have evaluated the performance of the ontology-based test generator as well as its capability of revealing faults on two case studies. The first case study (*Book-Trader*) is a book-trading MAS, similar to that summarized in Section 4. This system was implemented as a set of BDI agents [5] in JADEX [15]. We extended it to support ontology-based interaction. After modeling the interaction ontology (see Figure 2) using Protégé, we generated ontology-supporting code, and modified the implementation of *Seller* and *Buyer* agents accordingly. Moreover, we added OCL constraints (e.g., the price must be between 0 and 2000). The size of this MAS is 1312 Lines Of Code (LOC), 2 types of agent: buyer and seller.

For *BookTrader* we were able to obtain three ontologies with instances of books, comprising respectively 10, 20 and 100 instances. We applied diverse rules for valid and invalid input generation [13]. For instance, for the *Book* properties *author* and *title* we used the following three different rules for valid input: *(1) New value that has not been used before from ontology instances*; *(2) Reused value from ontology instances*; and *(3) Randomly generated value respecting rules in ontology.* Invalid inputs have been generated applying the two rules: *(1) Empty string*; and *(2) Randomly generated string.* Analogous rules (but for numeric datatypes), were used for generating valid and invalid inputs for the *Proposal*'s *price.*

Table 1 (a) shows the total number of test cases (divided into valid and invalid test cases) that were generated by executing continuous testing.Test case generation for the *Seller* required the creation of 3 test case templates, while only one template was needed for the *Buyer.* The small number of templates indicates that little manual effort is required by our approach. In fact, template definition is the only step requiring human involvement.

The two classes in the *BookTrader* ontology were fully covered by the automatically generated test cases. Moreover, two deviations from the expected behavior (faults) were observed. Manual testing of the same application was conducted by applying the goal-oriented test case derivation methodology described in our previous work [12]. Results are provided in Table 1 (b) and show that 6 test cases were manually defined for each agent under test. They cover the same number of classes in the ontology and reveal the same faults as the automatically generated test cases. Although in this example ontology-based test case generation exhibits no superior performance in terms of ontology coverage or fault detection, it increases the confidence in the correctness of the application, in that it allows exploring a much larger portion of the input space at no additional cost.

The second case study (*BibFinder*) is a MAS that aims at facilitating bibliographic search. A special assistant agent called *BibFinder* helps searching and building references for a specific topic, sharing bibliographic data with other *BibFinders*. In particular, *BibFinder* has the capability to:

1. Consolidate bibliographic data automatically, even when they are scattered geographically;
2. Perform searches on and extract bibliographic data from the Scientific Literature Digital Library[3], exploiting the Google search service[4];
3. Rank publications automatically based on usage history;
4. Form communities of *BibFinders* automatically in order to share bibliographic and ranking data of similar topics of interest;
5. Join an existing community or create a new community based on the interests of the *BibFinder*'s owner;
6. Recommend a list of "must-read" papers to the owner.

Similar to *BookTrader*, *BibFinder* is implemented as a BDI agent [5] in JADEX [15]. The size of this MAS is 8484 Lines Of Code (LOC), 3 agents. The main differences between *BookTrader* and *BibFinder* are that the former has been evolved together with the several versions of JADEX, hence it is likely to contain less faults than the latter, which was implemented recently from scratch. Moreover, the interaction ontology of *BibFinder* is much larger than *BookTrader*'s one.

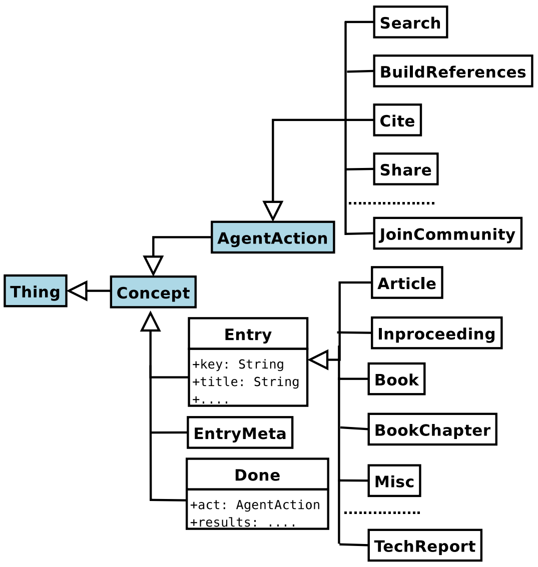


Fig. 3. Interaction ontology of *BibFinder*

A portion of the interaction ontology of *BibFinder* is presented in Figure 3. The complete ontology is quite big, because of the number of properties (e.g., `Entry` has 42 properties) and classes not shown in Figure 3 for space reasons (14 sub-classes of `Entry` and `AgentAction` are not shown in the figure).

[3] `http://citeseer.ist.psu.edu`

[4] `http://code.google.com/apis/soapsearch`

Table 1. Faults and coverage evaluation (N/A means Not Applicable)

MAS	**Agent under test**	**Numb of templates**	**Time limit**	**Number of test cases**			**Ontology coverage**	**Revealed faults**
				Valid	**Invalid**	**Total**		
BookTrader	*Seller*	3	30'	209	131	340	2/2	2
	Buyer	1	10'	56	38	94	2/2	2
BibFinder	*BibFinder*	5	1h	853	423	1267	27/27	6

(a) Automatically generated test cases

BookTrader	*Seller*	N/A	N/A	3	3	6	2/2	2
	Buyer	N/A	N/A	3	3	6	2/2	2
BibFinder	*BibFinder*	N/A	N/A	7	7	14	13/27	4

(b) Manually derived test cases

We used a set of BibTeX files, comprising a large number of BibTeX entries, to create a domain-specific ontology that specifies and contains BibTeX data. It was then aligned with the *BibFinder* ontology (shown in Figure 3). In total, we obtained 983 ontology instances.

Table 1 (a) shows the total number of test cases generated for *BibFinder*, ontology coverage and revealed faults. Compared to the manually derived test cases (Table 1, (b)), continuous, ontology-based testing allowed a much wider exploration of the input space, with a correspondingly higher ontology coverage and fault revealing capability. Reliability of *BibFinder* is substantially improved after the execution of continuous, ontology-based testing.

6 Conclusions

In this paper, we presented an experimental evaluation of a novel approach for automated test case generation using ontologies, which has been introduced in a companion paper [13]. The agent interaction ontology is combined with domain ontologies by means of ontology alignment techniques, so that domain ontology instances can be used to populate the agent interaction ontology with instances. Our test generator takes advantage of such instances to produce valid and invalid input messages that can be used to exercise the agents under test continuously.

Experimental results show that whenever the interaction ontology has non trivial size, the proposed method achieves a higher coverage of the ontology classes than manual test case derivation. It also overcomes manual derivation in terms of revealed faults, as well as portion of input space explored during testing. The level of automation achieved by our tool *eCAT* allows for test case generation at negligible extra costs.

The proposed approach focuses mainly on exploiting the ontology to test single agents, and it is limited in revealing faults resulting from single agent interactions. As a future work, we will extend the approach to test a team of agents, sharing a common ontology to reach team goals.

References

1. Bergenti, F., Gleizes, M.-P., Zambonelli, F.: Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Springer, Heidelberg (2004)

2. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous Agents and Multi-Agent Systems 12(2), 239–256 (2006)
3. Botía, J.A., Gómez-Sanz, J.J., Pavón, J.: Intelligent data analysis for the verification of multi-agent systems interactions. In: Corchado, E., Yin, H., Botti, V., Fyfe, C. (eds.) IDEAL 2006. LNCS, vol. 4224, pp. 1207–1214. Springer, Heidelberg (2006)
4. Botía, J.A., López-Acosta, A., Gómez-Skarmeta, A.F.: ACLAnalyser: A tool for debugging multi-agent systems. In: ECAI, pp. 967–968 (2004)
5. Bratman, M.E.: Intentions, Plans and Practical Reason. Harvard University Press, Cambridge (1987)
6. Coelho, R., Cirilo, E., Kulesza, U., von Staa, A., Rashid, A., Lucena, C.: Jat: A test automation framework for multi-agent systems. In: 23rd IEEE International Conference on Software Maintenance (2007)
7. Dikenelli, O., Erdur, R.C., Gumus, O.: Seagent: a platform for developing semantic web based multi agent systems. In: AAMAS 2005: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 1271–1272. ACM Press, New York (2005)
8. Euzenat, J., Bach, T.L., Barrasa, J., Bouquet, P., Bo, J.D., Dieng, R., Ehrig, M., Hauswirth, M., Jarrar, M., Lara, R., Maynard, D., Napoli, A., Stamou, G., Stuckenschmidt, H., Shvaiko, P., Tessaris, S., Acker, S.V., Zaihrayeu, I.: State of the art on ontology alignment. Knowledge Web Deliverable 2.2.3 (August 2004)
9. FIPA. Interaction protocols specifications (2000-2002), `http://www.fipa.org/repository/ips.php3`
10. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. The Knowledge Engineering Review 18(1), 1–31 (2003)
11. Lam, D.N., Barber, K.S.: Debugging Agent Behavior in an Implemented Agent System. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2004. LNCS, vol. 3346, pp. 104–125. Springer, Heidelberg (2005)
12. Nguyen, C.D., Perini, A., Tonella, P.: Automated continuous testing of multi-agent systems. In: The fifth European Workshop on Multi-Agent Systems (December 2007)
13. Nguyen, C.D., Perini, A., Tonella, P.: Ontology-based Test Generation for Multi Agent Systems. In: Proc. of the International Conference on Autonomous Agents and Multiagent Systems (2008)
14. Padgham, L., Winikoff, M.: Prometheus: A pragmatic methodology for engineering intelligent agents. In: Proc. Workshop on Agent Oriented Methodologies, OOPSLA 2002 (2002)
15. Pokahr, A., Braubach, L., Lamersdorf, W.: Multi-Agent Programming. In: Jadex: A BDI Reasoning Engine. Kluwer Book, Dordrecht (2005)
16. Rouff, C.: A test agent for testing agents and their communities. In: Aerospace Conference Proceedings, vol. 5 (2002)
17. TILAB. Java agent development framework, `http://jade.tilab.com/`
18. Tiryaki, A.M., Öztuna, S., Dikenelli, O., Erdur, R.C.: Sunit: A unit testing framework for test driven development of multi-agent systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE VII / AOSE 2006. LNCS, vol. 4405, pp. 156–173. Springer, Heidelberg (2007)
19. Zhang, Z., Thangarajah, J., Padgham, L.: Automated unit testing for agent systems. In: 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007), Barcelona, Spain (2007)

Testing and Debugging of MAS Interactions with INGENIAS

Jorge J. Gómez-Sanz[1], Juan Botía[2], Emilio Serrano[2], and Juan Pavón[1]

[1] Facultad de Informática, Universidad Complutense Madrid,
jjgomez,jpavon@sip.ucm.es

[2] Facultad de Informática, Universidad de Murcia,
juanbot,emilioserra@um.es

Abstract. Testing and debugging activities are getting more relevance in multi-agent systems (MAS) as agents become part of real applications. Both activities are related, since failures to be debugged are frequently detected during the execution of tests. The support for these activities is not yet as complete as other activities of MAS development. However, agent oriented software engineering methodologies are incorporating new testing and debugging features. In this direction, the paper introduces advances made in the INGENIAS agent development framework towards a complete coverage of testing and debugging activities. The advances are compared with respect to a categorisation of related works in the agent literature. This categorisation will be useful for evaluating and planning issues for improvement in the context of INGENIAS.

1 Introduction

Though development environments from MAS research improve year after year, testing and debugging aspects have not received the same attention [3]. Testing refers to the software development activities dealing with the detection of failures [17], while debugging refers to the detection and repair of the lines of code responsible of those failures. So far, there is an important amount of knowledge in conventional software engineering about testing and debugging, techniques which are being gradually applied to MAS [3]. Agent Oriented Methodologies are incorporating testing and debugging into their development process, like Passi [5] or Tropos [18]. To contribute to this effort, this paper introduces advances made in INGENIAS in the direction of testing and debugging.

INGENIAS has a distinguishing feature compared to most agent-oriented methodologies: its a model driven development approach [14]. It is not only using models, it is providing automatic code generation facilities, and mechanisms for synchronising changes in the code and the specification.

In INGENIAS, the implementation on a concrete target platform can be derived through a set of transformations from MAS design specifications. In order to cope with the complexity of MAS specification, each MAS is considered from five complementary viewpoints [15]: environment, agent, tasks and goals, organisation, and interaction.

Debugging in INGENIAS has been greatly improved with the incorporation of ACLAnalyser [4], a debugging tool for large scale MAS. Besides, the new facilities for debugging of the code generation framework, the INGENIAS Agent Framework, introduce new concepts for tracking the behaviour of the MAS visually and programatically. On the side of testing, the greatest advance has been the incorporation of test declarations at the specification level. The test coding is aided with tools for agent mental state inspection and the integration with the JUnit testing framework.

The inclusion of testing facilities has been addressed already in [5] or [18]. Nevertheless, testing proposed here differs in the primitives used to construct them. When writing a test, a developer refers to the existence or absence of entities appearing in the specification. To do so, the developer reuses entities generated automatically whose names match with those of the specification. Therefore, there is a clear mapping between the tests and what the specification. Also, there is not an urgent need to check that the code sticks to the specification, as in the previous mentioned approaches. After all, the code was automatically generated.

The paper introduces debugging facilities in INGENIAS in section 2. Then, section 3 describes testing facilities. These facilities are illustrated with an example in section 4. The related work can be reviewed in section 5. This is followed by the conclusions in the last section.

2 Debugging MAS in INGENIAS

INGENIAS provides basic debugging support through the INGENIAS Agent Framework, which facilitates the step by step execution of a MAS specification. This support has been recently complemented with the incorporation of ACLAnalyser [4] to deal with large scale MAS. ACLAnalyser is a software tool for debugging MAS that communicate using FIPA protocols. ACLAnalyser performs debugging by using data extracted from a real run of the MAS under development, and applying data mining techniques to the execution logs.

The integration consists in modifying the deployment of MAS and providing alternative protocol descriptions compatible with ACLAnalyser, by using INGENIAS code generation support. The deployment has been modified to include new launch scripts. These scripts replace references from JADE libraries to modified ones connecting with the ACLAnalyser. They also define specific ACLAnalyser execution parameters, like filtering the messages from the domain facilitator. With respect to the alternative protocol descriptions, they permit to evaluate if agents, during the execution, satisfy protocol specifications. These protocol specifications were integrated as new templates included in the code generation process. As a result, together with new launch scripts, the system produced specific protocol description files in XML format for the ACLAnalyser.

These features have been integrated in the *INGENIAS Analyser Module*, *IAM* from now on, a new module already incorporated to the main trunk of development of the INGENIAS Development Kit, IDK from now on. With the IAM and the incorporation of ACLAnalyser, a developer can do three different tasks. Firstly, visualising the communication links among agents. This representation shows a graph where the different agents being executed are nodes and the edges represent the existence of communicative acts among them. These edges are labelled with the total amount of bytes interchanged. Secondly, tracking individual conversations of defined protocols. Protocols different from the standard FIPA ones can be tracked so that a developer can tell if the conversations are progressing as expected. Finally, using causality graphs, a developer can trace back individual agent actions to determine what triggered a concrete behaviour in an agent.

Apart from the IAM, there is also some basic debugging infrastructure incorporated in the main code generation and execution framework included in the IDK, the INGENIAS Agent Framework, IAF from now on. As well as providing code generation facilities, the IAF performs static analysis of the specification, looking for missing specification elements and some semantic errors. Generated systems can be run with or without debugging facilities. Debugging facilities include the following services:

- Manual or automatic execution modes. This mode controls how agent tasks are executed. In manual mode, the user is shown all tasks scheduled for execution by all agents within the same Java Virtual Machine. When selecting a particular task, it is executed. In automatic mode, all scheduled tasks are executed automatically in the same order they were queued. Therefore, the user can decide whether to let the agent execute each scheduled task automatically or select manually which task to execute next. This permits to modify the execution order of tasks and to observe changes resulting from a single task execution.
- Representation of the current mental state. The IAF represents the information contained within the mental state of any agent of the current Java Virtual Machine. This information is presented following the same notation used to specify the system. As tasks are executed, the user can watch the changes. Usually, these changes happen too fast. However, in manual execution mode, all changes occur step by step, and always under the request of the user.
- Breakpoints at the task level. The user can select in the GUI the different task types an agent knows. After selecting, if the enable breakpoints option is activated, the system will switch to manual mode whenever a task of the selected type is going to be executed. This serves to let the system progress until the problematic task is going to be executed.
- Browsing current logs. This can be done during runtime or after the execution. In runtime, the IAF presents the different events organised according to their origin. Also, it can filter events and show those containing the string typed by the user. All logs are stored into files that can be inspected later on.
- Interaction protocol execution. The state of the execution of the protocol is represented using a state machine notation with labeled states. The developer can check at what point of the interaction protocol the failure happend

or even if the interaction protocol has finished or not. Logs for individual protocols (messages interchanged) are available as well.
– Dummy event generation. The events to be produced by the system, which feed the perception of each agent, can be triggered manually. This way, a developer can debug the MAS, generating those events that lead the system to failures or undesired behaviours.
– Mental entities traces. Each mental entity - like facts, goals, or events - stores information about who created it, who modified it, and who transmitted it. This way, it is possible to guess, whenever a failure is made, who provided the piece of information that led to the failure.

3 Testing MAS in INGENIAS

The INGENIAS methodology declares the existence of tests within the specification. This declaration is made by a developer and it is not produced automatically. In the current version, there are no means of defining the content of the test itself at the design level, either.

Though there are research lines in agents that study the automatic test generation, like [19], at this moment, INGENIAS follows a more conventional approach. Like in conventional software engineering [17], the tests are as good in detecting failures as the developer is. Hence, INGENIAS assumes there are qualified professionals who take the responsability of detecting critical situations in the system to be and design proper tests to meet these conditions.

In order to facilitate the specification of tests, the INGENIAS meta-model has been modified as shown in figure 1. A test definition has two parts: the declaration of the test and the declaration of the deployment. For the first, creating a test entity is enough. This entity has an identifier and a descriptor. For the second, it is necessary to define a *TestingPackage*. A *TestingPackage* defines the number and type of agents to be created, prior to the test. This information is captured by a *DeploymentPackage*. The definition of the number of instances is made within a *DeploymentUnitByType* entity. Specialisations of this entity permit an individual setup of each agent mental state so that the same type of agent can be run with different initial conditions, thought these extensions are not presented here. The *DeploymentPackage* and the *TestingPackage* can include special parameters to be used during the code generation stage (e.g. initial memory assigned and JADE communication port to be used).

In the implementation, a *Test* is realised as an extension of a JUnit class. JUnit (junit.sourceforge.net) is a widespread framework for unit testing in Java. The default test produced by the system contains a skeleton of code with examples of the kind of evaluations a developer can do. This skeleton is modified by the developer using several utilities. Among others, the IAF provides read/write access to the mental state of individual agents as well as some basic synchronisation primitives all of them with timeouts to prevent blocking calls.

TestingPackage is realised as an additional entry in the build and launch scripts, a JUnit suite class for running all the tests referred from the *Testing-Package*, and a JUnit specialisation of the JUnit class representing the test.

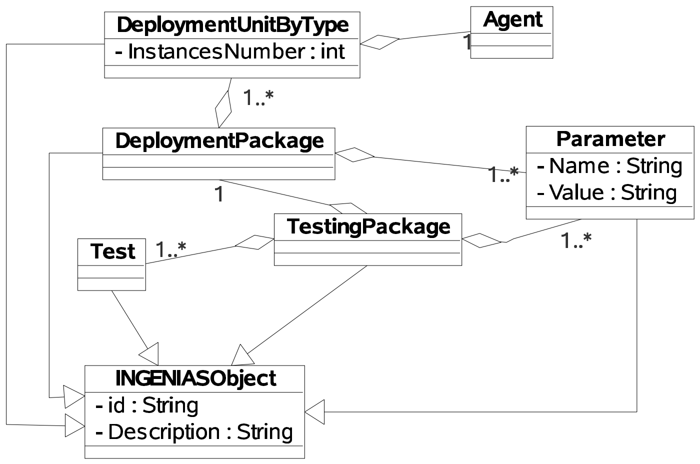


Fig. 1. Test infrastructure meta-model

This specialisation incorporates methods for initialising the system to be tested according to the *DeploymentPackage* referred within the *TestingPackage*.

The developer is supposed to elaborate the tests mainly observing the mental state of the agents and checking against the mental states depicted in the MAS specification. This approach permits as well to check interactions, since interactions in runtime are asserted in the mental state of INGENIAS agents as conversation entities. By accessing these entities and inspecting their values, a developer determines if the conversation among agents has finished or not, and what items of information they have exchanged.

The manipulation of mental states can be used as well to test conversations. This requires careful selection of the values of the mental entities so that they meet the conditions required to, first, launch the conversation, and, second, make the conversation progress towards the desired state.

The expected configuration of the mental state in runtime is defined in INGENIAS by means of the agent viewpoint. The agent viewpoint represents a collection of agent diagrams. These agent diagrams depict mental states to be reached along the execution of tasks and interactions. The mental state is an aggregation of mental entities such as goals, beliefs, facts, compromises, and others. These mental state representations are interleaved along the interactions in form of guards to be satisfied on delivering a message or on receiving it. Alternatively, a developer can choose to define agent diagrams representing mental states reached after a task execution. These diagrams do not have to declare all the entities of the mental state, only those that should exist together with boolean statements about the values they contain. The definition of tasks appear within the task goal viewpoint. These definitions include the list of expected inputs, outputs, and affected system components.

Another information source for test definition is the organisation viewpoint, which includes the definition of workflows and groups of agents. A workflow is a kind of view of the collaboration between two or more agents where tasks play a dominant role. Somehow, this view complements the protocol specification of an interaction as it appears in the interaction viewpoint.

4 An Example of Development with Testing and Debugging Capabilities

This section presents some of the new debugging capabilities of the IDK with a case study that is included in the IDK distribution, about a ticket selling service. This service provides user assistant agents, acting in representation of the user, which can contact cinemas for getting tickets for specific movies, and taking into account user characteristics. A user connects to the system and instructs a representative (User assistant agent) to obtain a cinema ticket. The assistant contacts another representative (buyer agent), which will be responsible for locating an appropriate cinema. The representative contacts different cinemas representatives (seller agent) until an adequate cinema is located. It is adequate when there are free seats and the price is good enough. Once obtained, the ticket is delivered, through the different representatives, to the user. The challenges in this case study are mainly the selection of a movie matching the interests of the user and selecting an appropriate cinema where this movie is available. The example will focus on the second aspect, which is more suitable for showing the interaction debugging capabilities.

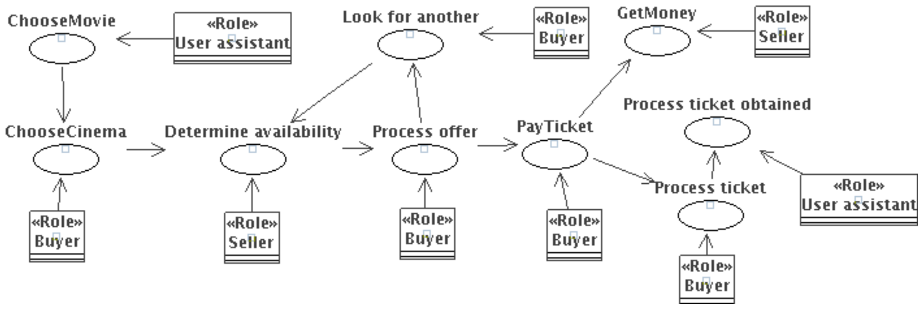


Fig. 2. Workflow defining the cinema ticket buying scenario

Initially, it was considered using a Contract-Net to solve the problem. However, the real situation discourages it. If a buyer made a call for proposals to the different cinemas, the cinema would be forced to block the offered seats. If enough buyers are considered, it may turn out that a subset of buyers block all the seats in all cinemas. This reduces the opportunities to let other buyers get seats in parallel. So this solution, while benefitting the buyer, makes cinemas lose money. An alternative to the solution is not to query all the cinemas at the same time, but going one by one.

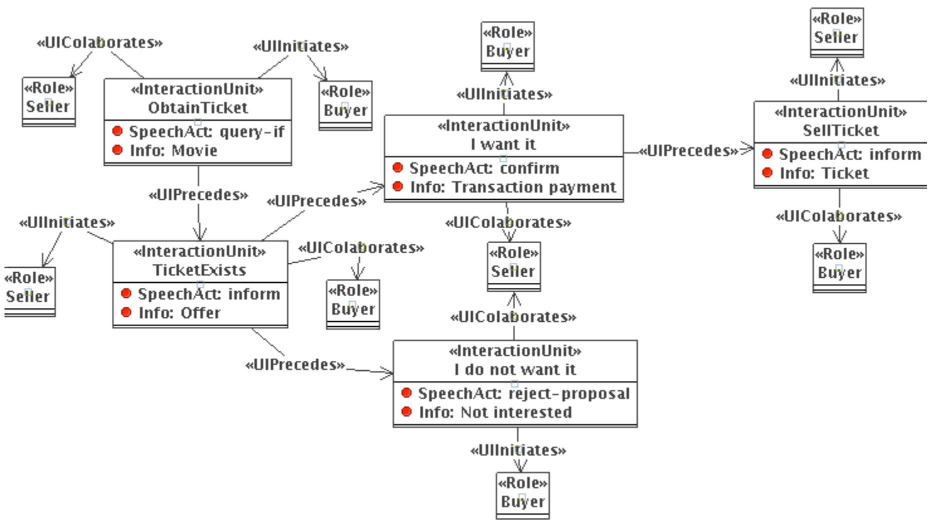


Fig. 3. Protocol for a buyer to obtain a ticket from seller

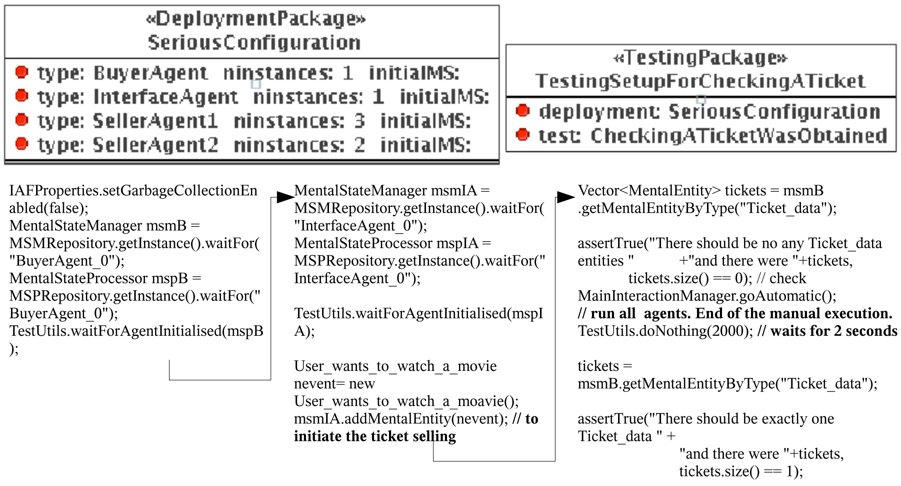
The workflow involved in the case study appears at the upper part of figure 2. This workflow starts with the user assistant selecting a movie and a candidate agent to act as buyer. The movie selection is passed to the chosen buyer so that it can start choosing a cinema to interact with. This cinema acts as a seller of tickets. The cinema determines if there are seats available and makes an offer. The Buyer decides if the offer is adequate or, if it is not, looks for another cinema. If it is adequate, it proceeds with the payment and delivers the ticket to the user.

A main interaction here is the dialogue between the buyer and the seller to obtain a cinema ticket. This dialogue is represented at the lower part of in figure 3. It uses interaction unit entities which represent the act of transferring information from one agent to another. The transferred information appears as info attributes while the speech act used follows FIPA conventions. Each interaction unit is initiated by the role supplying the information and continued by the role receiving the informations. The different interaction units are chained by means of precedence in time relationships. The sequence covers the information transfer required to execute part of the workflow represented with figure 2, concretely from task *ChooseCinema* to tasks *ProcessTicket* and *GetMoney*.

The scenario considers a deployment with one agent playing the role *buyer*, one agent playing the role *user assistant*, and five agents playing the role *seller*.

4.1 Testing the Scenario of the Example

To show how to test the system, a simple test is defined, see figure 4. The setup of the test requires one interface agent (User assistant), one buyer agent and five cinemas playing the seller role. The test to be applied is *CheckingATicketWasObtained*. The content of this test is presented on the right side of the figure. The test starts disabling the automatic garbage collection mechanisms to prevent entities to dissappear in the mental state. Not used entities are always removed in periodical checks of the system. Next, the mental state manager of the buyer and interface agents are acquired with two purposes. The first is gaining access to the mental state of these two agents. The second is synchronising the test with the lifecycle of the agents. To initiate the test, an event *User_Wants_to_watch_a_Movie* is asserted in the mental state of the interface agent. This event was generated automatically from the specification as a Java class. Next, the buyer is checked looking for the entity to be present when the buyer finishes its participation in the protocol from figure 3. The information *Ticket_data* is obtained as a result of the execution of task *Process ticket*, shown in figures 2. Before the system starts, this entity should not exist, and so it is asserted in the test. Following, the system is released from its manual execution mode (see section 2) so that all tasks are executed automatically. After 2 seconds, the buyer is expected to have the *Ticket_data entity.*


«DeploymentPackage»
SeriousConfiguration
type: BuyerAgent ninstances: 1 initialMS:
type: InterfaceAgent ninstances: 1 initialMS:
type: SellerAgent1 ninstances: 3 initialMS:
type: SellerAgent2 ninstances: 2 initialMS:
«TestingPackage»
TestingSetupForCheckingATicket
deployment: SeriousConfiguration
test: CheckingATicketWasObtained

```
IAFProperties.setGarbageCollectionEnabled(false);
MentalStateManager msmB =
MSMRepository.getInstance().waitFor("BuyerAgent_0");
MentalStateProcessor mspB =
MSPRepository.getInstance().waitFor("BuyerAgent_0");
TestUtils.waitForAgentInitialised(mspB);
```

```
MentalStateManager msmIA =
MSMRepository.getInstance().waitFor("InterfaceAgent_0");
MentalStateProcessor mspIA =
MSPRepository.getInstance().waitFor("InterfaceAgent_0");

TestUtils.waitForAgentInitialised(mspIA);

User_wants_to_watch_a_movie
nevent= new
User_wants_to_watch_a_moavie();
msmIA.addMentalEntity(nevent); // to initiate the ticket selling
```

```
Vector<MentalEntity> tickets = msmB
.getMentalEntityByType("Ticket_data");

assertTrue("There should be no any Ticket_data
entities "          +"and there were "+tickets,
          tickets.size() == 0); // check
MainInteractionManager.goAutomatic();
// run all  agents. End of the manual execution.
TestUtils.doNothing(2000); // waits for 2 seconds

tickets =
msmB.getMentalEntityByType("Ticket_data");

assertTrue("There should be exactly one
Ticket_data " +
                    "and there were "+tickets,
                    tickets.size() == 1);
```

Fig. 4. Example of the testing of a final state in the interaction protocol. At the top, the testing setup defined in the specification. At the bottom, the actual content of the test.

Further tests are defined to check if the selection of cinemas is appropriate. One should expect the user assistant agent to have some criteria about cinema selection. Hence, all cinemas should be consulted at some point in the execution. This is checked in the testing infrastructure as a consult to RuntimeConversation entities. These are entities contained within the mental state with references and data about interactions being executed. Hence, it is easy to collect traces of all executed interactions and ask if the number of cinemas asked matches the total population of the system. In the current development, this does not happen, as it will be seen in the next section.

4.2 A Debugging Session in INGENIAS

A first run of the system seems to work fine since the user sees in the user assistant agent, which is connected to a GUI, that the agent actually gets the tickets under the default preference parameters, concretely that the price of the ticket is under 3 euros and there is at least one seat available in the cinema.

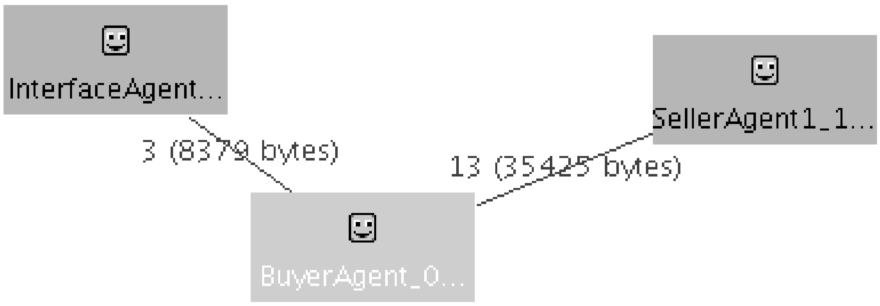


Fig. 5. ACLAnalyser communication graph indicating a coupling with only one cinema

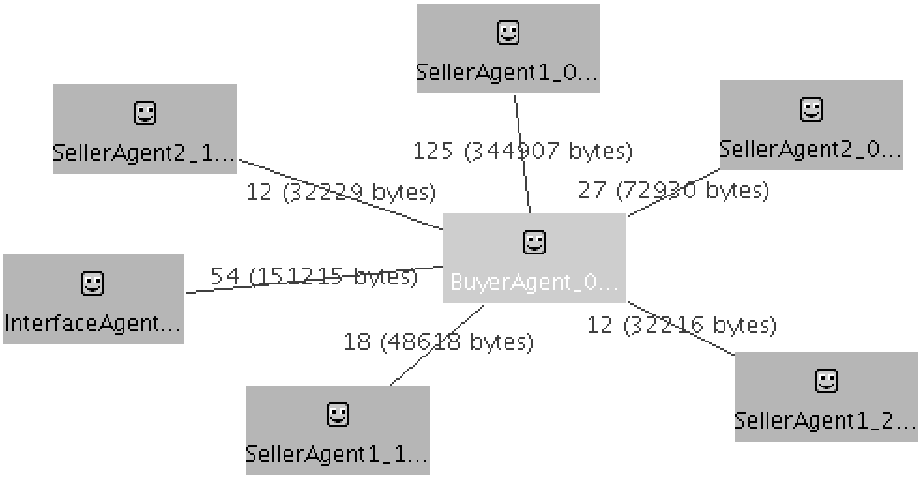


Fig. 6. ACLAnalyser communication graph showing a more relaxed coupling than in figure 5

To be sure, the IAM is run and specific debugging scripts are generated to connect generated instances with ACLAnalyser. Once launched, ACLAnalyser collects the statistics and presents the figure 5 . The graph shows a wrong result, since the agent seems to stick to only one cinema, concretely *SellerAgent1_1*, while it should be showing communications with other cinemas as well. After modifying the cinema selection task, ChooseCinema task in the figure 2, the result is the figure 6. This time, all cinema agents, the five of them, seem to be in touch with the buyer agent. This fits better in the image of an agent looking for an offer among existing cinemas. The debugging can go further by using the clustering capabilities of the ACLAnalyser. For instance, one may expect, if trust models are applied, that after a time, the agent engages with a subset of existing cinemas.

5 Related Work

This section presents research works in the field of testing and debugging of MAS software, which have been categorised following the three levels that are considered by Ferber [13]: agent, group, and society.

5.1 The Agent Level

Literature includes many examples of testing single agents, almost all of them influenced by the BDI (*Belief, Desire, Intention*) model of agency. For example, the Tracer Tool [16] uses beliefs, goals, intentions, actions, events, messages, and relations between them to create a concept graph. This concept graph defines the behaviour model that the agent must accomplish. Sudeikat et al. [25] considers beliefs, goals and plans in the context of JADEX agents platform. Their approach focuses on checking, by means of assertions and static checking, the consistency of beliefs, the correct adoption of goals and appropriate plan execution. AgentFactory [8] offers syntactic checking at compile time and some debugging facilities at run time by using the Agent viewing tool through which it is possible to analyser the agent's internal state and control execution steps. An alternative approach, which does not rely on the BDI model is that of Coelho et al. [7]. In this work, agents are seen as black boxes. Testing is performed by sending concrete messages to agents and checking if responses for individual messages are correct.

Visual debugging and testing facilities at the agent level in INGENIAS are similar to those in [8], for mental state inspection and testing. There are also facilities to visualise interactions. Furthermore, code generation performs static checks on the specification. However, the automatic generation of models to check agent behaviour, as in [16] remains as future work. Validation of agents is not addressed as in [7]. Since the system generates automatically all agents, it does not make sense to produce additional mock agents to simulate the interaction: it is already done.

5.2 The Group Level

Testing at the group level consists in testing coordination issues derived from interaction protocol (IP) based conversations.

At the moment, there are few references in the literature that properly address group testing. Well known works are those done in the context of the

Prometheus methodology [21,22] although they check mainly the correct accomplishment of conversations following clearly defined interaction protocols. A different approach is that of Alberti et al. [1]. This work follows a declarative approach and takes advantage of the semantics of communicative acts to verify some social integrity constraints. More works in the line of interaction protocols verification are [11,23,20]. These works are three examples which show how to convert some diagrammatic and informal specification of any IP into a formal one, allowing to model any IP in a general way, to analyse and verify it later. Another approach to model and analyse some IP is proposed in [12]. In this case an IP is analysed by capturing events causality occurrence in the MAS and obtaining a diagram that shows event causality. By means of an IP pattern library and performing pattern recognition, interactions occurred in the MAS are analysed to improve them.

INGENIAS is not applying formal approaches yet to verify the system behaviour. It relies on the expertise of developers to define a collection of tests that is sufficient to validate the group level behaviour. In INGENIAS, testing at this level would imply checking that workflows, like the one presented in the figure 2, are executed. This can be programmed in a test by asking each individual agent about the pieces of information being asserted into individual mental states across different agents. Also, the IAM would help here with the observation of agent protocols with the ACLAnalyser.

5.3 The Society Level

Testing a MAS as a society refers to check if some properties, restrictions and/or rules are accomplished during the life time of the society. Hence, the scope of such properties, restrictions, and rules is the whole system, not individual or group components. Properties may be related with quality features the system must observe.

The importance of quality features in MAS is considered in [26]. Examples of these quality features are flexibility, manageability, response time, or openness. Others may make sense depending on the domain. For instance, a MAS of agents in an electronic market would use a global wealth measurement to track MAS evolution. An alternative way to look at the society implies that, instead of tracking the accomplishment of properties, we look at emergent properties which were not specified in the design. This is specially interesting for the multi-agent based simulation community [9]. It is of interest, for example, to detect emergent behaviours in the society which were not included in the specification of a group of agents in the general MAS design [10]. Also, emergent behavior can imply direct influence in the individual agents as individual beliefs can be affected [6].

The testing and debugging facilities are considered in part in INGENIAS. Response time is easily tested by the implicit definition of timeouts in the tests, just like the wait 2 seconds command from figure 4. Also, since INGENIAS can define arbitrary deployment configurations for testing, the developer can choose the workload of the system. Emergence of properties is harder to define and test. It would imply defining in the test what values are expected in the agents after some simulation time. Part of this emergence detection could be aided with the ACLAnalyser. For instance, the creation of communities of agents with strong interaction links can be detected with the clustering facilities of the ACLAnalyser. Nevertheless, the society level testing and debugging requires more work.

6 Conclusions

The paper has introduced advances in testing and debugging made in the INGENIAS methodology and implemented on IDK. In this context, we have integrated the functionality provided by a stand alone tool for debugging MAS software, ACLAnalyser, into a MAS development framework, the IDK. Also, the support for testing and debugging of IDK have been introduced.

The scope of the work presented reaches the agent, group, and society testing levels, though the later in a lower degree. Testing is based on the assertion of the mental state of individual agents. This assertion covers as well the interaction execution, since its state is recorded into the mental state of its participants. This permits to address testing at all levels, though the society level requires more effort.

The transference of the presented results to other methodologies is not trivial. A methodology willing to reuse these results needs to satisfy three requirements. First, the target methodology needs to be strongly based on meta-models and have a design tool that takes advantage of them, otherwise, the test meta-model reuse would not make sense. Second, the reuse of the IAM requires being able to process protocol descriptions to produce ACL Analyser configuration files. Though most methodologies do use protocol descriptions, not all of them provide facilities to explore the specification and produce other files. Third, the reuse of the ACL Analyser itself requires using JADE as the target platform. Despite the general purpose of existing methodologies, implementation stage tends to focus on concrete agent platforms, not always JADE. For instance, Prometheus rely on Jack agent platform, which is not compatible with the ACL Analyser.

Acknowledgements

This work has been supported by the project *Methods and tools for agent-based modelling* supported by Spanish Council for Science and Technology with grants TIN2005-08501-C03-01 and TIN-2005-08501-C03-02, by the project *Análisis, Estudio y Desarrollo de Sistemas Inteligentes y Servicios Telemáticos* through the Fundación Séneca within the Program *Generación del Conocimiento Científico de Excelencia*, and by the grant for Research Group 921354 by the Region of Madrid (Comunidad de Madrid) and the Universidad Complutense Madrid.

References

1. Alberti, M., Daolio, D., Torroni, P., Gavanelli, M., Lamma, E., Mello, P.: Specification and verification of agent interaction protocols in a logic-based system. In: SAC 2004: Proceedings of the ACM symposium on Applied computing, pp. 72–78. ACM, New York (2004)
2. Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.): PROMAS 2006. LNCS, vol. 4411. Springer, Heidelberg (2007)
3. Bordini, R.H., Braubach, L., Dastani, M., El Fallah Seghrouchni, A., Leite, J., Gómez-Sanz, J.J., O'Hare, G., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica Journal 30(1), 33–44 (2006)
4. Botía, J.A., Hernansaez, J.M., Gómez-Skarmeta, A.F.: Towards an approach for debugging mas through the analysis of acl messages. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS, vol. 3187, pp. 301–312. Springer, Heidelberg (2004)
5. Caire, G., Cossentino, M., Negri, A., Poggi, A., Turci, P.: Multi-agent systems implementation and testing. In: From Agent Theory to Agent Implementation Fourth International Symposium (AT2AI-4), Vienna, Austria (2004)
6. Castelfranchi, C.: Simulating with cognitive agents: The importance of cognitive emergence. In: Sichman, et al. (eds.) [24], pp. 26–44
7. Coelho, R., Kulesza, U., von Staa, A., Lucena, C.: Unit testing in multi-agent systems using mock agents and aspects. In: SELMAS 2006: Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems, pp. 83–90. ACM, New York (2006)
8. Collier, R.W.: Debugging agents in agent factory. In: Bordini, et al. (eds.) [2], pp. 229–248
9. Conte, R., Gilbert, N., Sichman, J.S.: Mas and social simulation: A suitable sommitment. In: Sichman, et al. (eds.) [24], pp. 1–9.
10. David, N., Sichman, J.S., Coelho, H.: Towards an emergence-driven software process for agent-based simulation. In: Sichman, J.S., Bousquet, F., Davidsson, P. (eds.) MABS 2002. LNCS, vol. 2581, pp. 89–104. Springer, Heidelberg (2003)
11. Fadil, H., Koning, J.-L.: Rules for translating interaction protocols into a b formal representation. In: Skowron, A., Barthès, J.-P.A., Jain, L.C., Sun, R., Morizet-Mahoudeaux, P., Liu, J., Zhong, N. (eds.) IAT, pp. 495–498. IEEE Computer Society, Los Alamitos (2005)
12. El Fallah-Seghrouchni, A., Haddad, S., Mazouzi, H.: A formal study of interactions in multi-agent systems. I. J. Comput. Appl. 8(1) (2001)
13. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
14. Fuentes-Fernández, R., Gómez-Sanz, J.J., Pavón, J.: Model driven development of multi-agent systems with repositories of social patterns. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 126–142. Springer, Heidelberg (2007)
15. Gómez-Sanz, J.J., Pavón, J.: Methodologies for developing multi-agent systems. Journal of Universal Computer Science 10(4), 359–374 (2004)
16. Lam, D.N., Barber, K.S.: Comprehending agent software. In: AAMAS 2005: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, pp. 586–593. ACM, New York (2005)

17. Myers, G.J., Sandler, C., Badgett, T., Thomas, T.M.: The Art of Software Testing, 2nd edn. Wiley, Chichester (2004)
18. Nguyen, C.D., Perini, A., Tonella, P.: A goal-oriented software testing methodology. In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 58–72. Springer, Heidelberg (2008)
19. Nguyen, D.C., Perini, A., Tonella, P.: ecat: a tool for automating test cases generation and execution in testing multi-agent systems. In: AAMAS (Demos), IFAAMAS, pp. 1669–1670. (2008)
20. Paurobally, S., Cunningham, J., Jennings, N.R.: Developing agent interaction protocols using graphical and logical methodologies. In: Dastani, M., Dix, J., El Fallah-Seghrouchni, A. (eds.) PROMAS 2003. LNCS, vol. 3067, pp. 149–168. Springer, Heidelberg (2004)
21. Poutakidis, D., Padgham, L., Winikoff, M.: Debugging multi-agent systems using design artifacts: the case of interaction protocols. In: AAMAS, pp. 960–967. ACM, New York (2002)
22. Poutakidis, D., Padgham, L., Winikoff, M.: An exploration of bugs and debugging in multi-agent systems. In: AAMAS 2003: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pp. 1100–1101. ACM, New York (2003)
23. Quenum, J.G., Aknine, S., Briot, J.-P., Honiden, S.: A modeling framework for generic agent interaction protocols. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS, vol. 4327, pp. 207–224. Springer, Heidelberg (2006)
24. Sichman, J.S., Conte, R., Gilbert, N. (eds.): MABS 1998. LNCS, vol. 1534, pp. 1–9. Springer, Heidelberg (1998)
25. Sudeikat, J., Braubach, L., Pokahr, A., Lamersdorf, W., Renz, W.: Validation of bdi agents. In: Bordini, et al. (eds.) [2], pp. 185–200
26. Weyns, D., Schelfthout, K., Holvoet, T.: Architectural design of a distributed application with autonomic quality requirements. SIGSOFT Softw. Eng. Notes 30(4), 1–7 (2005)

PASSI Methodology in the Design of Software Framework: A Study Case of the Passenger Transportation Enterprise

Daniel Cabrera-Paniagua and Claudio Cubillos

Pontificia Universidad Católica de Valparaíso, Escuela de Ingeniería Informática, Av. Brasil 2241, Valparaíso, Chile
daniel.cabrerap@gmail.com, claudio.cubillos@ucv.cl

Abstract. This paper presents a practical experience on the use of the PASSI methodology in conjunction with a general framework development process, in obtaining a software framework for a virtual enterprise for passenger transportation. In addition to background information on each of the topics discussed, diverse PASSI artifacts complemented with notational elements drawn from UML-F are shown. In addition, the experience on the use of PASSI for framework development is provided.

Keywords: PASSI, Framework Development Process, Agents, Passenger Transportation.

1 Introduction

The agent paradigm constitutes a significant forward step in the future of systems development, similar to a revolution in the software area [5]. Until a few years ago, the development of multiagent systems considered very few the use of software engineering techniques [9]. The development of these systems was based on ad hoc procedures, which allowed a high degree of flexibility to the characteristics of the project tackled. However, there were innumerable problems, which mainly are summarized in deficiencies of utilization of available resources, and precarious levels of quality, since the formal processes of testing and quality assurance did not exist. Therein lays the importance of using software engineering techniques in the systems development, and in this particular case, its application to the development of multiagent systems. While this offers various direct benefits, it should be noted that the level of the final obtained result depends, among others, on the quality of the software development process used.

The present work describes the application experience with PASSI methodology [1] in the design of a software framework for the domain of passenger transportation. Because the project involved a software framework, it has been considered the use of a general process for framework development along with PASSI. Additionally, we have incorporated some elements offered by a non-standard UML profile called UML-F [7], devoted to the development of object-oriented software frameworks.

The novelty of our work relies on: 1) show the practical experience of using PASSI in obtaining a software framework and 2) An analysis of the evidence obtained through the study case.

2 Related Work

In some European countries local authorities have introduced flexible public transport and demand responsive services that have proved to be most popular among users, and thus support the retention of citizens in their every day environment. Pilot experiences for DRT systems were developed during projects such as SAMPO [10], SIPTS [11], and FAMS [13]. Regarding examples of MAS applied to ITS, just to mention some initiatives, Ferreira et al. [14] presented a multi-agent decentralized strategy where each agent was in charge of managing the signals of an intersection and optimized an index based on its local state and "opinions" coming from adjacent agents. In 2002, Cai and Song [12] introduced a traffic control model with MAS, in which a more flexible agent self-control framework was described and a multi-agent negotiating strategy was conceived.

This work represents the continuity of a past research in this transportation domain [6] [17], concerning the development of an agent system for passenger transportation for a single operator under a demand-responsive scenario.

3 The Framework Development Process

As mentioned above, the methodology to use for developing software is of vital importance. Here are reviewed those aspects relating to software development process adopted in the present work.

3.1 The PASSI Methodology

PASSI (Process for Agent Societies Specification and Implementation) is a step-by-step methodology for designing and developing multiagent systems. PASSI integrates design models and concepts from both OO software engineering and artificial intelligence approaches using the UML notation. Figure 1 shows the PASSI methodology, which is made up of five models plus twelve steps in the process of building a multi-agent system. For a more detailed description on the different steps please refer to [1]. According to the figure, and considering as an exception the phase of "Agent Implementation Model", highlights the prominent sequentiality of the methodology. This contrasts with the iterative nature of the development of software frameworks. That is why PASSI has been used within an overall process of software framework development.

3.2 Process of Developing Software Frameworks

Historically, one of the most important topics taken into account in the area of software development is its reuse. Software reuse allows reaching a faster software development while promising a higher quality level. In this sense, software

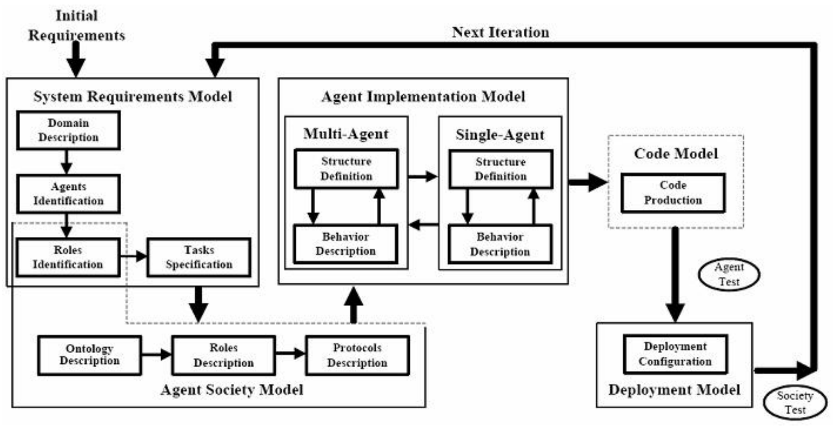


Fig. 1. The PASSI methodology

frameworks, one of the alternatives to carry out reuse, have gained considerable popularity in both the industry and academia. According to [8], *"a framework is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuses at a larger granularity than classes"*.

Certainly, the software systems development is not an easy task, given mainly by the large number of variables and constraints involved. And in this respect, to develop reusable software system in time is much more difficult. A flexible software system must meet its requirements, while confining the solution for a wide range of future problems. That is why it is necessary to have the maximum possible aid elements. One is the process of developing frameworks, in a systematic way that will get a higher quality framework.

In the literature exists several proposals about development process for the obtaining of software frameworks [2] [19]. The considered process for framework development corresponds to the one presented in [3], where some stages are identified that should be part of an overall process of software framework development (see Figure 2). The steps involved in this approach are [3]:

Analysis of the problem domain. This is performed systematically or through development of one or a few applications in the domain and the key abstractions are identified. The first version of the framework is developed utilizing the key abstractions found.

One or possibly a few applications are developed based on the framework. This is the testing activity of the framework. Testing a framework to see if it is reusable is the same activity as developing an application based on the framework. Problems when using the framework in the development of the applications are captured and solved in the next version of the framework. After repeating this cycle a number of times the framework has reached an acceptable maturity level and can be released for multi-user reuse in the organization. Is important to say that the success of development process of a framework depends on the

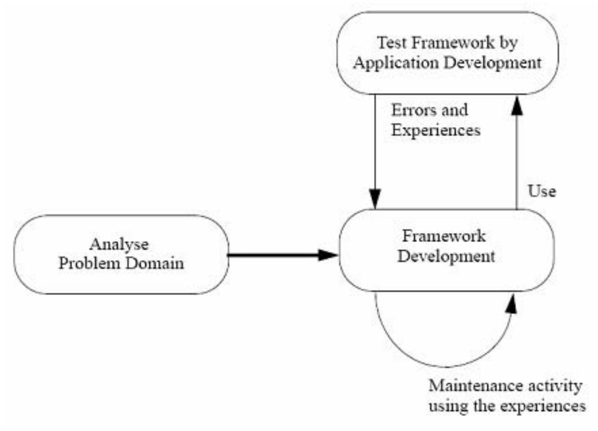

Test Framework by Application Development
Errors and Experiences
Use
Analyse Problem Domain
Framework Development
Maintenance activity using the experiences

Fig. 2. The General Framework Development Process

experience of the organization in the problem domain the framework addresses. A more experienced organization can select a more advanced development process since they will have fewer problems with the problem domain.

The software frameworks raise that it is possible achieve reuse, through the development of software configurable architectures, based primarily on identifying points of fixed nature, and points of variability. Determine the points of the architecture that should be variable and the points that are not is the most difficult task in developing software frameworks. So, the general framework process development mentioned previously indicates, as a basis, an iterative development process, considering the experience and feedback to the fullest possible way. Hence it is mentioned that a software framework never ends developing it. Over time, sooner or later it is necessary to make modifications, to a greater or lesser degree, to initial architecture reached.

Particularly at present work, the main objective consists in development an agent architecture for a specific problem domain, the passenger transportation enterprise. But must not lose sight of that the architecture should offer a flexible capacity, in the sense of dispose obtaining several final systems from it. In other words, is necessary achieve a software framework for the specific problem domain addressed. For this reason, the framework design is supported by the use of two processes as a whole. On the one hand, there is a general framework development process, which roughly indicates the overall development line, namely: an initial analysis of the problem domain, actors and systems involved, and so on; development of the framework itself (obtaining the software artifacts) and validation, through an application built by extending the framework.

On the other hand, lays PASSI, a multiagent systems development methodology, aiding the development of complex, multi-party, distribute and heterogeneous software systems based in the agent concept as base modeling unit.

3.3 The UML-F Profile

The UML-F profile represents an important alternative in the development of software frameworks, because it formalizes aspects not covered by the UML standard. Some of the most important features of the UML-F profile are: to provide elements of notation to adequately document well-known design patterns; is built on the standard UML, that is, the extensions generated can be defined on the basis of extension mechanisms which in UML already exist; is, in itself, the medium that allows a direct way to document any pattern framework. The PASSI methodology was designed for the development of agents systems and not precisely for the developing of agent-oriented software frameworks. For the same reason, the UML-F profile was used in the development of some artifacts, with the aim to fill this gap. Please refer to [7] for more details on the UML-F notation.

4 The Study Case: Passenger Transportation Enterprise

The Intelligent Transport Systems (ITS) have been attracting interest of the transport professionals, the automotive industry and governments around the world. The ITS aim at the development of the road infrastructure (for example, ways) and to integrate them together with the persons and vehicles by means of advanced technologies of integration, from several research areas.

Regarding the public transport domain, in the last years the Demand Responsive Transport (DRT) services have risen in popularity. A DRT is understandable as a component of a long chain of inter modal service, delivering local and complementary mobility to other conventional transport means, such as fixed-line buses and trains. However, geographical coverage problems among transport operators services, difference in the volume and quality of handled information and, in general, a fragmentation in the transport service provision, gives origin to problems that range from wrong evaluations coming from state-regulatory entities, up to direct problems with the system final users, which definitively results in a poor quality of service. For these reasons in many cases the solution implies a better integration and coordination of the diverse parties, leveraging the concept of virtual enterprise.

4.1 Virtual Enterprise

A virtual enterprise is a cooperation network of legally independent companies, which are quickly united and mainly contributed their basic competences in sequence to exploit a specific business opportunity. In general terms, the life cycle of a virtual enterprise is marked by four phases, that go from the identification, evaluation and selection of business opportunities, to the selection of partners to conform the virtual enterprise; later, a phase of operation, in where the business opportunity is exploit; and a phase associated at the end of the virtual enterprise, with the corresponding separation of assets. In general, virtual enterprises are applicable to all those domains where it is possible to conceive a collaboration of different companies or entities, taking as a benchmark to reach certain goals for themselves while using information technologies. In this sense, we may think in a virtual transportation enterprise, with an increased level of adaptability to the offer, considered the variability in the levels of existing demand. With this, and under new business opportunities, the virtual transportation enterprise adapts its structure to meet the existing demand.

4.2 Transport Requirements

In a complex system like the passengers transportation one, there are many users or actors who have a direct interest in the commercial, social and infrastructure impacts. The actors considered in a DRT service correspond to:

User: Represents the end user of the passengers transport system. The user has the faculty to make request of transport (with its respective conditions), as well as to indicate some problem that affects to him and that has incidence in the concretion of the trip requested.

Transport Operator: Represents a transport company within the system. A transport operator can correspond to an only person (even handling to she herself a vehicle, without delegating that responsibility in a conductor), or also correspond to a company composed by multiple vehicles (fleet of vehicles). The virtual transportation enterprise is conformed by manifold operators.

VE Administrator: The Virtual Enterprise Administrator represents the central administration of the virtual transportation enterprise. Its faculties are related to affiliation control of new transport operators into the virtual transportation enterprise and its permanence, and taking action when extern events to virtual transportation enterprise happen, without restricting the future allocation of other responsibilities.

Driver: This is a vehicle driver belonging to a transport operator. Receives the itinerary to follow, and can notify problems or indications to meet your itinerary (for example, report a delay in the time of encounter with a passenger).

Government Entity: Is a governmental organization with regulating or control faculties, which guard current legislation and that service contracts are fulfilled.

Active Destination: Represents a frequent destiny within total set of existing destinations. An active destination can make the virtual transportation enterprise see a necessity or a business opportunity available; as well as indicate problems associated to the same transport service, like a loss in the quality of service, or restrictions on the operation.

Traffic Information System: Represents an external information system that gives information on present traffic conditions, collisions, among others.

VE - Customers Manager System: Controls the profile of each user of the virtual transportation enterprise. The life of each user (related to the virtual transportation enterprise) is recorded in the first instance, with purely operational purposes, which does not prevent in the future can use this information for strategic purposes.

VE - Transaction System: This system controls all transportation requests completed, the requests are under way, and even those that have been canceled for various reasons, including all information existing in the request for transport, and the service characteristics offered by the virtual transportation enterprise.

VE - Affiliates Enterprises Management System: This system manages the life cycle of the virtual enterprise, from transport operators incorporation until the separation of transport operator from the virtual transportation enterprise.

TO - Fleet Management System: This system is responsible for managing the vehicles that comprise the transport operator fleet. This system depends directly on the transport operator.

TO Solver: This system has the task to optimize transportation operations (planning and scheduling) using a solver and/or heuristic software, on which are scheduled trips to perform for each vehicle.

5 Multiagent Framework Design

In this section, the agent framework artifacts are depicted following the PASSI steps. The first diagram presented shows a portion from the Agent Identification Diagram (see Figure 3), which is framed within the first stage of the PASSI methodology, corresponding to the System Requirements Model.

This diagram takes as starting point the description of UML use-cases, offering a general view of all the functionality provided by the system and in addition, it incorporates a grouping of use-cases for each agent identified within the system in order to visualize the responsibility level that each of the agents has regarding the system. The generation of diagrams is given on the basis of the use of a graphical tool available for PASSI, called PASSI Toolkit [4].

The UserAgent is who represents within the system the interests of the transport service user. It administers the transport preferences user, and manages his service requests. For this reason, it establishes communication with the

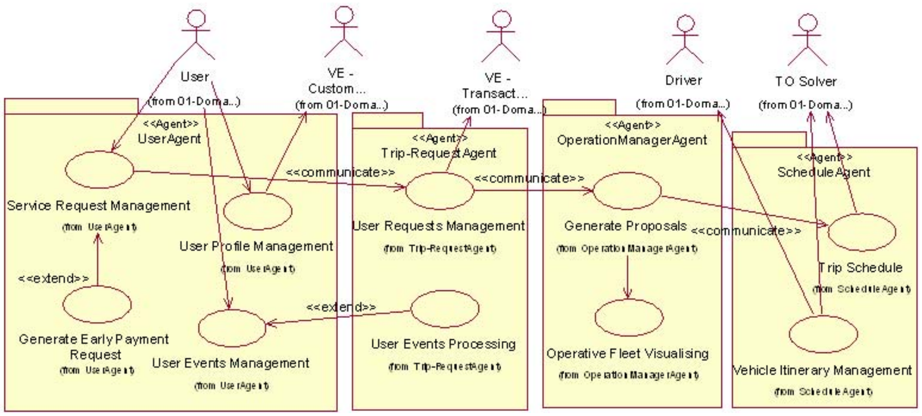


Fig. 3. Agent Identification Diagram

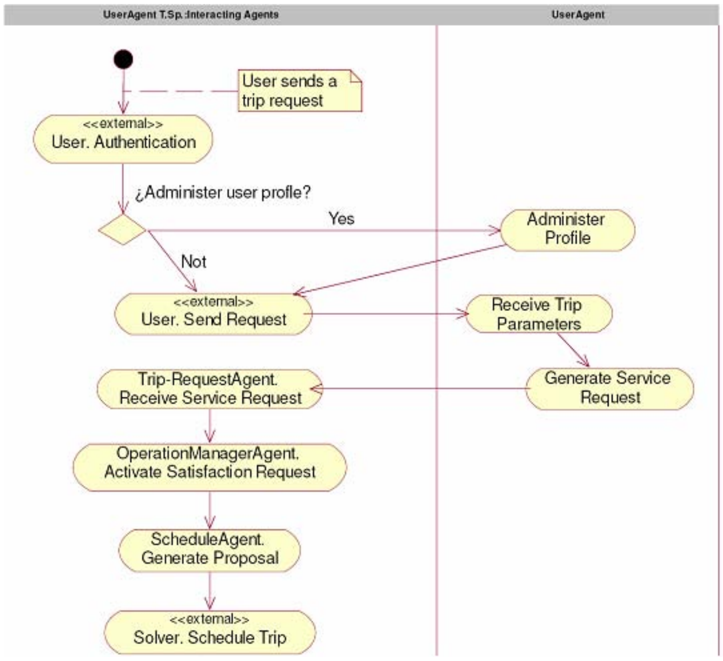

UserAgent T.Sp.:Interacting Agents
UserAgent
User sends a trip request
<<external>> User. Authentication
¿Administer user profle?
Yes
Not
Administer Profile
<<external>> User. Send Request
Receive Trip Parameters
Generate Service Request
Trip-RequestAgent. Receive Service Request
OperationManagerAgent. Activate Satisfaction Request
ScheduleAgent. Generate Proposal
<<external>> Solver. Schedule Trip

Fig. 4. Task Specification of the UserAgent

Trip-RequestAgent. Also, it allows the user the generation of an advanced-payment request, considering for this several means to carry it out. It can also communicate problems (called "events"), for example, report as late for meeting with the transport vehicle assigned for the trip. Each event is recorded in a single list for each user, and is sent to PlannerAgent for administration.

The Trip-RequestAgent manages the request of trips emitted by the user of the transport service, and maintains a registry of the pending requests. It sends the requests for its processing to the OperatorManagerAgent, receiving the proposals generated for each conducted request. Later, is recorded in the VE - Transactions System, the received request and the vehicle identifier.

The OperatorManagerAgent receives users trip requests, and active mechanisms for transport operators affiliated to the virtual enterprise attempt to generate an offer to the request. For this, knows the vehicles available at all times, and in operational service.

The ScheduleAgent verifies for a particular vehicle if it fulfills the conditions specified on a requested trip (user conditions, conditions of the virtual enterprise, or conditions caused by external events), checking its itinerary obtained from the information system of the transport operator. Considering the feasibility verification, a proposal or a declination takes place.

Figure 4 shows a portion from Task Specification Diagram for the agent UserAgent, where a user of the transportation system sends a service request.

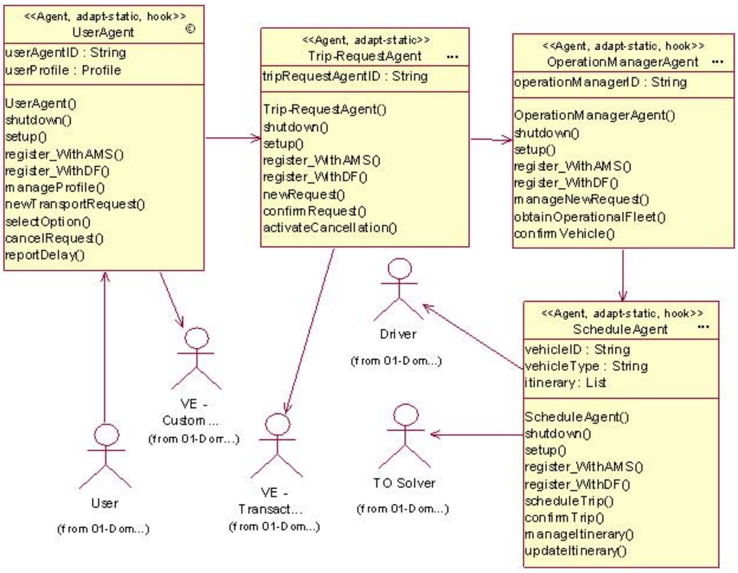


Fig. 5. Multiagent Structure Definition Diagram

The send of a transport request involves indicating a set of parameters on the trip, which are clustered: places (origin and destination), schedules (hour meeting at the place of origin, and arrival time at destination), and trip preferences. These preferences are managed through a user profile, which can vary over time. After having details of the travel request, it is received by the agent OperationManagerAgent, which establishes the operational fleet and makes a broad call for sending proposals to meet the request. Each vehicle (represented by the agent ScheduleAgent) tries to schedule the trip requested within their itinerary.

The Figure 5 shows a portion from diagram of the phase of Agent Implementation Model,which is the Multiagent Structure Definition. It is possible to view all actors within the defined architecture in development, and its relationship with the various agents, and the identifying the transactions related to each of them. The classes identified in the figure with the symbol "...", indicates that have not yet been established all internal elements of them (both attributes and methods). On the other hand, the classes with the symbol "©" are those that their methods and attributes shown are actually all the ones the class possesses. The stereotype <<agent>> indicates that the classes are agents, and the stereotype <<adap-static>> denotes those classes that may be subject to changes, but only changes at the design phase (at runtime is not possible to observe changes in its internal structure). The stereotype <<hook>> indicates that the class has at least one method of type "hot spot", that is, their characteristics depends on each particular implementation derived from the model defined.

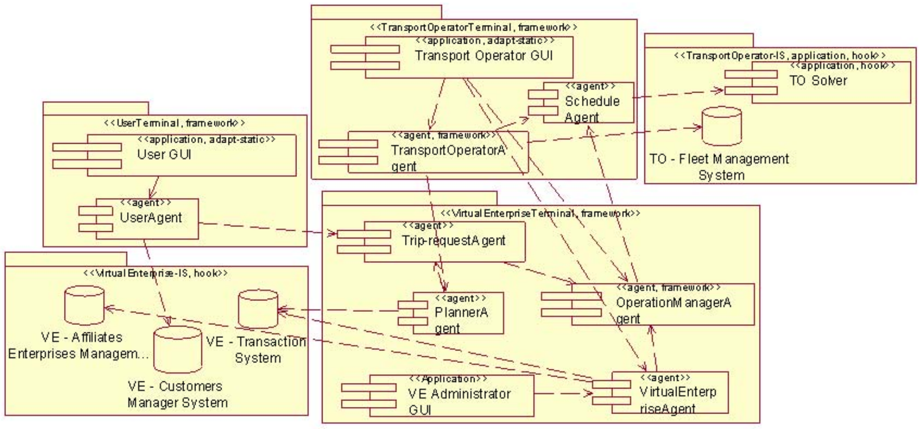

<<TransportOperatorTerminal, framework>>
<<application, adapt-static>>
Transport Operator GUI
<<agent>>
Schedule Agent
<<agent, framework>>
TransportOperatorAgent
<<TransportOperator-IS, application, hook>>
<<application, hook>>
TO Solver
TO - Fleet Management System
<<UserTerminal, framework>>
<<application, adapt-static>>
User GUI
<<agent>>
UserAgent
<<VirtualEnterpriseTerminal, framework>>
<<agent>>
Trip-requestAgent
<<agent>>
PlannerAgent
<<agent, framework>>
OperationManagerAgent
<<Application>>
VE Administrator GUI
<<agent>>
VirtualEnterpriseAgent
<<VirtualEnterprise-IS, hook>>
VE - Affiliates Enterprises Managem...
VE - Transaction System
VE - Customers Manager System

Fig. 6. Deployment/Component hybrid diagram of the framework

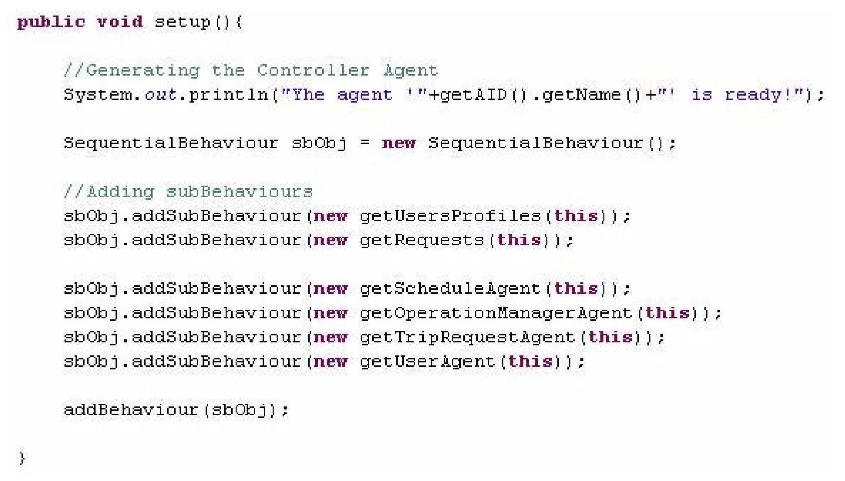
Next, a deployment/component hybrid diagram shows a portion of the general framework architecture (see Figure 6). Some packages have the UML-F stereotype <<framework>>. This means that these components are owned solely to the framework architecture. On the other hand, there are some packages that have the stereotype <<application>>. This stereotype indicates that these elements do not belong directly to the framework architecture, but are external components to framework, related somehow to it.

This architecture allows each transport operator affiliated to the virtual transportation enterprise to control at any moment the status of its operative fleets, as well as to administer all the information of their own information systems, in such a way that independence between the different transport operators stays in the operative scope. Each transport operator has his own mechanism for allocation and control of the itineraries for the different vehicles conforming its fleet, having the virtual transportation enterprise the only responsibility of receiving trip requests and the assignment of these requests to the vehicle that constituted the most attractive provider for the service user.

The Figure 7 shows a portion code of the UserAgent agent. This code is part of a functional prototype that is currently under development and is being tested with Solomons benchmark data sets for VRP and specially adapted for the passenger transportation problem.

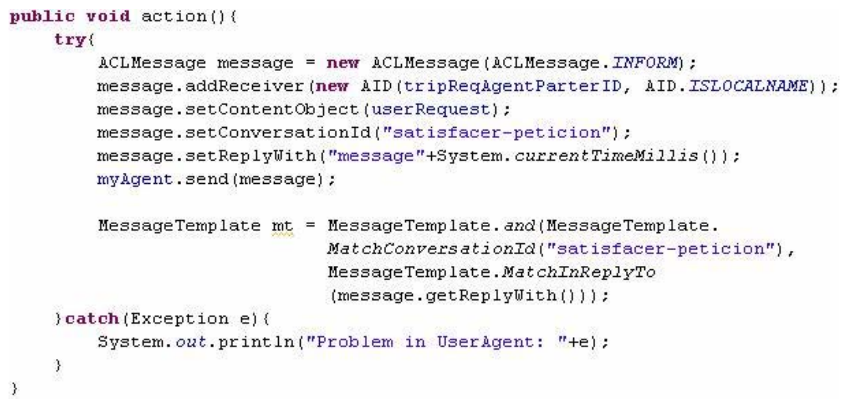
The objective sought to develop a functional prototype is get some measure of how architecture raised behaves in a context closer to reality, so that in the future decide to develop applications from it, in a real context. The development of a functional prototype of large-scale, also demand efforts and resources to the same extent. That is why it is considered a limited scenario in the functional prototype.

The agents considered in the prototype for those who are directly involved in the receipt and administration of transport requests: UserAgent, ScheduleAgent,



```
public void setup(){

    //Generating the Controller Agent
    System.out.println("The agent '"+getAID().getName()+"' is ready!");

    SequentialBehaviour sbObj = new SequentialBehaviour();

    //Adding subBehaviours
    sbObj.addSubBehaviour(new getUsersProfiles(this));
    sbObj.addSubBehaviour(new getRequests(this));

    sbObj.addSubBehaviour(new getScheduleAgent(this));
    sbObj.addSubBehaviour(new getOperationManagerAgent(this));
    sbObj.addSubBehaviour(new getTripRequestAgent(this));
    sbObj.addSubBehaviour(new getUserAgent(this));

    addBehaviour(sbObj);

}
```

Fig. 7. Code from the Controller Agent

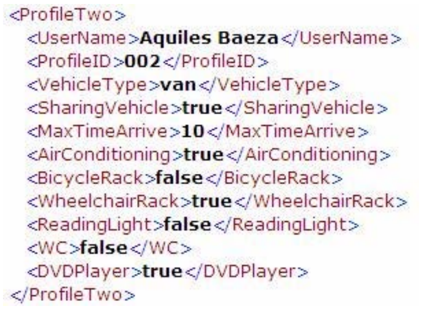


```
public void action(){
    try{
        ACLMessage message = new ACLMessage(ACLMessage.INFORM);
        message.addReceiver(new AID(tripReqAgentParterID, AID.ISLOCALNAME));
        message.setContentObject(userRequest);
        message.setConversationId("satisfacer-peticion");
        message.setReplyWith("message"+System.currentTimeMillis());
        myAgent.send(message);

        MessageTemplate mt = MessageTemplate.and(MessageTemplate.
                             MatchConversationId("satisfacer-peticion"),
                             MessageTemplate.MatchInReplyTo
                             (message.getReplyWith()));
    }catch(Exception e){
        System.out.println("Problem in UserAgent: "+e);
    }
}
```

Fig. 8. Code from the UserAgent agent

Trip-RequestAgent, and OperationManagerAgent. In addition to the previously agents, an agent is incorporated during the prototype performance. This agent is called Controller (see Figure 8).

The Controller Agent performs certain tasks: Get the User Profile of each user's transportation system considered within the prototype; obtain transportation requests of each of the users of transport system (from an XML file); obtain the properties of each vehicles considered within the prototype (from an XML file), and generate the instances of agent ScheduleAgent with such information; generate the instances of agent UserAgent, including both the User Profile as the transportation request associated with each user; and finally, generate the instances of the remaining agents.



```
<ProfileTwo>
  <UserName>Aquiles Baeza</UserName>
  <ProfileID>002</ProfileID>
  <VehicleType>van</VehicleType>
  <SharingVehicle>true</SharingVehicle>
  <MaxTimeArrive>10</MaxTimeArrive>
  <AirConditioning>true</AirConditioning>
  <BicycleRack>false</BicycleRack>
  <WheelchairRack>true</WheelchairRack>
  <ReadingLight>false</ReadingLight>
  <WC>false</WC>
  <DVDPlayer>true</DVDPlayer>
</ProfileTwo>
```

Fig. 9. XML Code from an user profile

The prototype code being developed by Jade Platform [16]. The Jade platform offers some assurances of stability for applications developed with its technology, because it has the backing of major companies, and is implemented on Java technology, an industry technology consolidated around the world.

The information from each user profile, applications for transport, vehicles and properties, are obtained by the use of XML files (see Figure 9). The feed prototype based on XML files replaced the inclusion of real people and systems that deliver this information in a real context.

The prototype takes as its bases the work developed by Cubillos et al., which is based in a work developed in the year 1986 by Jaw et al. [18].

As mentioned, the scope of the current prototype is limited. Mainly, they are looking to get some kind of feedback on the overall architecture raised, as well as the entities included in it, and business processes defined. In this sense, has completed implementation of all actors involved in the scenario described, by subtracting complete the development of the component called TO Solver, associated with the node TransportOperator-IS. At present, prototype does not incorporate a mechanism for scheduling the requests of trip received to the various vehicles available within the virtual transportation enterprise, because the component TO Solver is currently in development.

6 Analyzing PASSI

Based on the experience with the PASSI methodology and from the present study case, it is possible to make some analysis on PASSI appropriateness. First, its use provides a traceability of requirements along its phases, through the obtaining of its various artifacts. This allows easily identify and isolate any problems that exist within the system models, and the same way, more accurately verify that the requirements tackled in the early stages of the methodology are adequately represented in the final solution.

On the other hand, the early identification of actors, agents and functionality associated with each agent is of vital importance, since it allows the work team to have an overview but comprehensive system development. Fundamentally, this is reflected in having clarity about which agents will be integrated into the system, without needing to think about elements to be addressed in the future, as is its implementation technology. The tasks assignment to each agent complements the above mentioned, giving an overview of the features that are the responsibility of each agent.

The multiagent structure definition diagram describes agent classes involved in the system. These classes have a direct relationship with the agents identified in the first phase of PASSI. At this level, it is possible to observe attributes and behaviors in each agent, which gives a closer view to the final implementation. Automatic generation of this diagram through the PASSI Toolkit contributes significantly to the maintenance of consistency during systems development. In view of the above, the PASSI methodology constitutes an important guide in the development of systems based on agent technology. However, it also suffers from some elements that, if incorporated, would significantly improve the PASSI adoption in industry and academia. For example, it does not engage the user explicitly within the development process, for example, the inclusion of the final users from the initial steps, in order to guide the development. In the domain tackled in the present work, this has relevance, because the passenger transport domain considers direct participation of various human actors with the information system. Therefore, it becomes necessary to incorporate the user, for example, in some stage of usability tests on user interfaces.

Also, although PASSI incorporates the "testing idea" (both individual agent and overall level of the multiagent society), it does not explicit how carry them out. This lack of guidance in the testing reveals perhaps more important, as reflects the general lack of quality assurance processes. Anyway, it is necessary to mention that the Agile PASSI methodology [15] incorporates formally this step into the development process.

On the same line, a possible future work is to develop a proposal of process for the development of multiagent systems, but incorporating project management tasks, e.g. risk management, quality assurance, among others. With this, we can achieve a major improvement in agent software projects at industrial level, while complementing its existing level of use in the academic environment.

Finally, as regards the framework development and PASSI, it is possible to say that it is necessary to adapt PASSI at various points, in order to accommodate the iterative nature of the framework development process. Although it is iterative, the period of time required to complete a full PASSI iteration requires considerable effort.

Therefore, on some occasions it was necessary to return quickly to any past stage of PASSI, in order to correct and stabilize some requirements not entirely clear, breaking this sequentiality established by PASSI. Although this action have a cost (because stops the process, apply the necessary improvements, and finally, check the consistency between the different diagrams), it is better than waiting until the last steps of development process and start a new iteration just to begin to apply the corrections needed. For example on some occasions different problems were discovered in the step of Roles Identification and was necessary to return directly to Agent Identification Diagram, breaking the natural sequentiality of PASSI.

7 Conclusions

A practical experience on the use of the PASSI methodology in conjunction with a general framework development process has been achieved. An analysis on the results obtained from the use of PASSI has been exposed. It is worth noting that PASSI is not a complete development methodology of software projects, it lacks certain stages and some generic software project artifacts. Future work is devoted principally to gather more background on the advantages and challenges offered by PASSI methodology to be used in obtaining agent-oriented software frameworks, in order to make any specific proposal to extent the obtained results.

Acknowledgments

This work has been partially funded by the Pontifical Catholic University of Valparaíso (www.pucv.cl) through project No. 037.215/2008 "Collaborative Systems" Nucleus Project and by CONICYT through FONDECYT research grant No 11080284.

References

1. Burrafato, P., Cossentino, M.: Designing a multiagent solution for a bookstore with the passi methodology. In: Fourth International Bi-Conference Workshop on AgentOriented Information Systems, AOIS 2002 (2002)
2. Johnson, R.: How to Design Frameworks. In: Tutorial Notes, 8th Conference on Object-Oriented Programming Systems, Languages and Applications, Washington, USA (1993)
3. Mattsson, M.: Object-Oriented Frameworks: A Survey of Methodological Issues. Technical Report 96-167, Dept. of Software Eng. and Computer Science, University of Karlskrona/Ronneby
4. PASSI Toolkit (PTK), `http://sourceforge.net/projects/ptk`
5. Jennings, N.: On agent-based software engineering. Artificial Intelligence 117, 277–296 (2000)
6. Cubillos, C., Guidi-Polanco, F.: An Agent Solution to Flexible Planning and Scheduling of Passenger Trips. IFIP AI 217, 355–364 (2006)
7. Fontoura, M., Pree, W., Rumpe, B.: The UML Profile Framework Architectures. Addison Wesley, Reading (2000)
8. Johnson, R., Foote, B.: Designing reusable classes. Journal of Object-Oriented Programming 1(2), 22–35 (1988)
9. Gomez, J.: Metodologías para el diseño de sistemas multiagente. Revista Iberoamericana de Inteligencia Artificial 18, 51–64 (2003)
10. SAMPO TR1046 - Systems for Advanced Management of Public Transport Operations, `http://www.cordis.lu/telematics/tap_transport/research/projects/sampo.html`

11. SIPTS - TEN45607 - Services for Intelligent Public Transport Systems, `http://www.novacall.fi/sipts/e_default.htm`
12. Cai, Z.H., Song, J.Y.: Model of Road Traffic Flow Control based on Multi-agent. Journal of Highway and Transportation Research and Development 19(2), 105–109 (2002)
13. FAMS - Flexible Agency for Collective Demand Responsive Services. IST-2001-34347, `http://www.famsweb.com`
14. Ferreira, E.D., Subrahmanian, E.: Intelligent Agens in Decentralized Traffic Control. In: IEEE Intelligent Transportation Systems Conference Proceedings, USA, August 2001, pp. 705–709 (2001)
15. Chella, A., Cossentino, M., Sabatucci, L., Seidita, V.: Agile PASSI: An agile process for designing agents. Journal of Computer Systems: Science and Engineering 21(2) (2006)
16. JADE: Java Agent Development Framework, `http://jade.tilab.com`
17. Cubillos, C., Guidi-Polanco, F., Demartini, C.: Towards a Virtual Enterprise for Passenger Transportation Using Agents. In: Sixth IFIP Working Conference on Virtual Enterprises, Valencia, Spain, vol. 186, pp. 569–576 (2005) ISBN 978-0-387-28259-6
18. Jaw, J., Odoni, A.R., Psaraftis, H.N., Wilson, N.H.M.: A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with time windows. Transportation Research B 20B, 243–257 (1986)
19. Wilson, D., Wilson, S.: Writing frameworks - capturing your expertise about a problem domain. In: Tutorial notes, The 8th Conference on Object-Oriented Programming Systems, Languages and Applications, Washington (1993)

Developing and Evolving a Multi-agent System Product Line: An Exploratory Study

Ingrid Nunes[1], Camila Nunes[1], Uirá Kulesza[2], and Carlos Lucena[1]

[1] PUC-Rio, Computer Science Department, LES
Rio de Janeiro - Brazil
{ioliveira,cnunes,lucena}@inf.puc-rio.br
[2] New University of Lisbon
Lisboa - Portugal
uira@di.fct.pt

Abstract. Software Product Line (SPL) approaches motivate the development and implementation of a flexible and adaptable architecture to enable software reuse in organizations. The SPL architecture addresses a set of common and variable features of a family of products. Based on this architecture, products can be derived in a systematic way. A multi-agent system product line (MAS-PL) defines a SPL architecture, whose design and implementation is accomplished using software agents to address its common and variable features. This paper presents the evolutionary development of a MAS-PL from an existing web-based system. The MAS-PL architecture developed is composed of: (i) the core architecture represented by the web-based system that addresses the main mandatory features; and (ii) a set of software agents that extends the core architecture to introduce in the web system new optional and alternative autonomous behavior features. We report several lessons learned from this exploratory study of definition of a MAS-PL.

Keywords: Software product lines, multi-agent systems, object-oriented design, aspect-oriented programming.

1 Introduction

Software engineering aims to produce methods, techniques and tools to develop software systems with high levels of quality and productivity. Software reuse is one of the main strategies proposed to address these software engineering aims. Software reuse techniques provide many benefits, such as reduction of development costs and time to market, and quality enhancement. Over the last years, many reuse techniques have been proposed and refined by the software engineering community. Examples of these techniques are: component-based development, object-oriented (OO) application frameworks and libraries, software architectures and patterns. One of the latest trends in software reuse is the product line approach.

Software product lines [26,6] (SPLs) refer to engineering techniques for creating similar software systems from a shared set of software assets using a systematic method in order to build applications. Clements & Northrop [6] define a software product line (SPL) as “a set of software intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”. The main idea of SPL engineering is to develop a reusable infrastructure that supports the software development of a family of products. A family of products is a set of systems that has some commonalities, but also variable features. According to [7], a feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a product line.

Over the last years, agent-oriented software engineering (AOSE) has also emerged as a new software engineering paradigm to allow the development of distributed complex applications which are characterized by a system composed of many interrelated subsystems [18]. Most of the current AOSE methodologies are dedicated to developing single multi-agent systems [25]. New approaches [24,9] have started to explore the adoption of SPL techniques to the context of multi-agent systems (MASs) development. The aim of these new approaches is to integrate SPL and AOSE techniques by incorporating their respective benefits and helping the industrial exploitation of agent technology. Nevertheless, there are still many challenges to overcome in the development of multi-agent systems product lines (MAS-PLs) [25].

In this work, we relate our experience of development of a MAS-PL in a bottom-up fashion. Our MAS-PL has been developed and emerged from the evolution of a conference management web-based system. Each new version of the system includes the design and implementation of new features that the previous version does not address. Most of the new features are related to the introduction of autonomous behavior in the original system using MAS technology, such as agents, roles and their associate behaviors. All the three versions of our MAS-PL share a common SPL core architecture. The purpose of this case study was to create a scenario to identify problems related to the integration of the SPL and MAS technologies. Some of the challenges that we are trying to deal with during the development of the MAS-PL were: (i) how to modularize agency features in a MAS-PL; (ii) how to seamlessly incorporate autonomous behavior (agency features) in a traditional web-based system; (iii) which SPL models are useful and essential to specify and document an MAS-PL, and how to adapt them to the context of MAS-PL development. We also discuss possible solutions to these problems, such as adopting aspect-oriented programming (AOP) to isolate existing agency features.

This remainder of this paper is organized as follows. Some works related to multi-agent systems and product lines are described in Section 2. The Expert-Committee MAS Product Line is presented in Section 3. The discussions and lessons learned are presented in Section 4. Finally, the conclusions and directions for future works are discussed in Section 5.

2 Multi-agent Systems and Product Lines

Some recent research work has investigated the integration synergy of Multi-Agent Systems (MASs) and Software Product Lines (SPLs) technologies. Dehlinger & Lutz [9] have proposed an extensible agent-oriented requirements specification template for distributed systems that supports safe reuse. Their proposal adopts a product line to promote reuse in multi agent systems, which was developed using the Gaia methodology. The requirements are documented in two schemas: the role schema and the role variation point. The first defines a role and the variation points that a role can play during its lifetime. The second captures the requirements of role variation points capabilities. The proposed approach allows the reuse of agent configuration along the system evolution. Each agent configuration can be dynamically changed and reused in similar applications.

Pena et al [24] describe an approach to describing, understanding, and analyzing evolving systems. The approach is based on viewing different instances of a system as it evolves as different "products" in a software product line. Following their approach, a SPL is developed with an AOSE methodology, and the system is viewed as a MAS-PL. Their approach proposes a set of software engineering techniques based on an agent-oriented methodology, called Methodology for analyzing Complex Multi-Agent Systems (MaCMAS), whose main aim is to deal with complex unpredictable systems. The MaCMAS allows for the description of the same feature at different levels of abstraction. It presents a process to building the core architecture of a MAS-PL at the domain engineering stage, whose activities are: (i) to build an acquaintance organization; (ii) to build a feature model; and (iii) to analyze commonalities and to compose core features. The main advantage of the approach resides in the fact that it makes it possible to derive a formal model of the system and each state that it may reach.

Our work also explores the combination of MAS and SPL techniques and technologies in the context of development and evolution of systems. We focus specifically in the web-based systems domain by proposing the introduction of autonomous behavior features inside traditional web layered architectures. Our main aim was to investigate and understand the benefits of the agency feature modularization during this process.

3 The ExpertCommittee MAS Product Line

This section describes our experience in the development of a multi-agent system product line (MAS-PL) for the web domain. Our experience is presented along the next sections in a stepwise fashion. We initially present the ExpertCommittee (EC), a web-based conference management system that supports the paper submission and reviewing processes from a conference (Section 3.1). After that we describe an evolved version of this system in which we have incorporated new agency optional and alternative features related mainly to the specification and implementation of user agents (Section 3.2). This new version of the EC system is characterized as a multi-agent system product line (MAS-PL), because it addresses a new set of optional and alternative agency features which allows providing different customized versions of the system. The EC MAS-PL architecture designed to address the new agency features is then presented in terms of the components and agents that compose the system (Section 3.3). Finally, we detail the OO design and implementation of the EC MAS-PL by describing the mechanisms adopted to implement its variabilities (Section 3.4).

3.1 The ExpertCommittee Web-Based System

The ExpertCommittee (EC) is a typical web-based application whose aim is to manage the paper submission and reviewing processes from conferences and workshops. The EC system provides functionalities to support the complete process of conference management, such as: (i) create conferences; (ii) define conference basic data, program committee, areas of interest and deadlines; (iii) choose areas of interest; (iv) submit paper; (v) assign papers to be reviewed; (vi) accept/reject to review a paper; (vii) review paper; (viii) accept / reject paper; (ix) notify authors about the paper review; and (x) submit camera ready. Each of these functionalities can be executed by an appropriate user of the system, such as, conference chair, coordinator, program committee members and authors. Figure 1(a) shows the feature model [7] of the first version of EC system. This version was considered the core of our MAS-PL, created with the incorporation of new optional features in the subsequent versions.

The EC web-based system was structured according to the Layer architectural pattern [11] and is composed of the following components/layers: (i) `GUI` - this layer is responsible to process the web requests submitted by the system users. It was implemented using the Struts[1] framework; (ii) `Business` - is responsible to structure and organize the business services provided by the EC system. The transaction management of the business services was implemented using the mechanisms provided by the Spring[2] framework; and (iii) `Data` - aggregates the classes of database access of the system, which was implemented using the Data Access Object (DAO) design pattern. The Hibernate[3] framework was used to make persistent the objects in a MySQL[4] database. Figure 2 illustrates the architecture of the EC web-based system and highlights the core architecture.

The first implemented version of our EC system is a common web application that has the features mentioned above. In the following versions, software agents were introduced on the EC system, adding autonomous behavior. We consider autonomous behavior actions that the system automatically performs and previously needed human intervention. Examples of features provided by the agents are the deadline monitoring and tasks management. We also added a new role, the reviewer, which can review a paper delegated by a committee member. Next sections detail these new versions and respective features present in their implementations.

[1] http://struts.apache.org/
[2] http://www.springframework.org/
[3] http://www.hibernate.org/
[4] http://www.mysql.org/

3.2 Evolving the EC System to an MAS-PL

There are different SPLs adoption strategies [20]. The proactive approach motivates the development of product lines considering all the products in the foreseeable horizon. A complete set of artifacts to address the product line is developed from scratch. In the extractive approach, a SPL is developed starting from existing software systems. Common and variable features are extracted from these systems to derive an initial version of the SPL. Finally, the reactive approach advocates the incremental development of SPLs. Initially, the SPL artifacts address only a few products. When there is a demand to incorporate new requirements or products, the common and variable artifacts are incrementally extended in reaction to them.

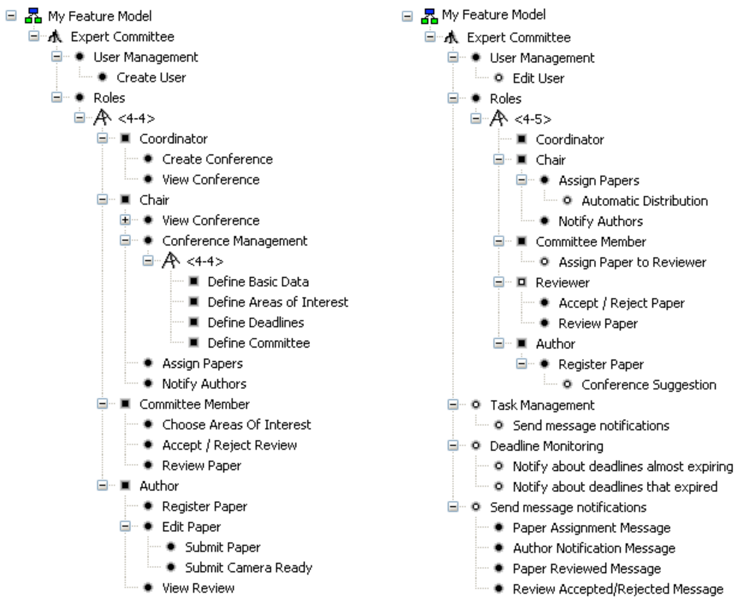
Our case study was developed considering the reactive approach. We have evolved the original version of the EC System (Section 3.1) to incorporate new optional and alternative features mainly related to autonomous behavior. The main aim of these new features was to help the tasks assigned to all the system users by giving them a user agent that addresses the following functionalities: (i) deadline and pending tasks monitoring; and (ii) automation (or semi-automation) of the user activities. In the third version of the EC system, we improved the modularization of many of these optional and alternative agency features to enable their automatic (un)plugging into the original system.

Table 1 summarizes the three different versions of our EC system. The first version was built without any autonomous behavior, in other words, without software agents. It was detailed in Section 3.1. The second version of the EC system contains features that are related to autonomous behavior and it has also some new features that add functionalities to the system as well. This version

Table 1. The three versions of ExpertCommittee

Version	**Description**
Version 1	Typical web-based application that represents our MAS-PL core. It has the mandatory features that support the conference management process. These features are described in Section 3.1.
Version 2	Release 1: Addition of the Reviewer role and the functionalities related to it: accept/reject review and review paper. Release 2: Addition of automatic suggestion of conferences to the authors. Release 3: Addition of message notifications to the system users through email or SMS (alternative feature). Release 4: Addition of deadline monitoring, to trigger specific actions when they expire. Release 5: Addition of automatic assignment of papers to committee members review them. Release 6: Addition of task management.
Version 3	Refactoring of Version 2 to improve the modularization of some agency features in order to make possible the automatic product derivation.

was developed in six different releases, each of them addressing a new optional or alternative feature. Each new release was implemented based on the previous one. The software agent abstraction was used to model and implement the autonomous behavior presented by the new agency features. A software agent is an abstraction that enjoys mainly the following properties [29]: autonomy, reactivity, pro-activeness and social ability. Thus, in this second version, we have used the agent abstraction and AOSE techniques to allow the introduction of new optional and alternative agency features in the system. Figure 1(b) illustrates the feature model containing the new agency optional and alternative features introduced in the second version of the EC system.



(a) Mandatory Features of the EC (b) Optional Features of the EC

Fig. 1. Expert Committee Feature Model

The third and last version of the EC system was implemented by applying a series of refactorings in version 2. The system was restructured to make the (un)plugging of optional features possible. Each optional feature was modularized by using a combination of OO design patterns and techniques with Spring configuration files that allows the injection of dependencies inside the variable points of the EC SPL architecture. It improves the capacity to produce and compose different configurations (products) of the SPL, and it also enables the automatic product derivation by means of model-based tools, such as: software factories [16], generative programming [7], GenArch [5,4], pure::variants [27]. Product derivation is the process of constructing a product from the set of assets specified or implemented for a SPL [8]. Each product is composed of the

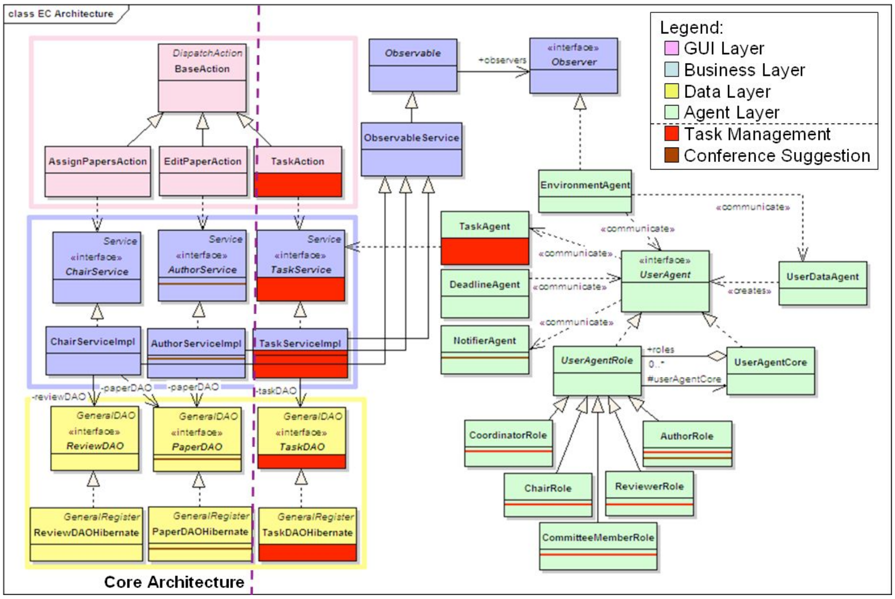


Fig. 2. MAS-PL Architecture

core features and a valid combination of optional and alternative features, according to the feature model. In an automatic product derivation process, the application engineer can generate a configuration (product) of the SPL by only selecting and choosing the features that are going to compose your product.

3.3 The EC MAS-PL Architecture

The EC Version 2 was implemented as an SPL architecture, which is illustrated in Figure 2. New features associated with the autonomous behavior of the system were added as a set of optional features. Different software agents and agent roles were specified to modularize these features. The JADE[5] framework was used as the base platform to implement our agents. These agents are responsible for monitoring the execution of different functionalities of the EC in order to provide their respective functionalities. The integration between the web architecture and the agents was accomplished by means of the introduction of the Observer pattern [12]. All the services that make part of the Business layer extends the `Observable` class. This class has a set of objects that implement the `Observer` interface. The `EnvironmentAgent` implements this interface, and is notified about changes in the system. Details about each agent that comprises the system are listed below:

[5] http://jade.tilab.com/

Environment Agent: this agent monitors the EC system by observing the execution of specific business services. These monitored events of the EC system represent the environment in which the user agents are situated. Each user agent is specified to perceive changes in the environment and make actions according to them. The environment agent was implemented using the Observer design pattern [12]. When it is initialized, it registers itself as an observer of the services that compose the `Business` layer. These services are observable objects that allow the observation of their actions. That means that, for each call of the system business methods, the services not only execute the requested methods, but they also notify their respective observers. The only observer in our implementation is the `EnvironmentAgent`, whose aim is to notify the other agents of the MAS-PL about the system changes;

User Data Agent: this agent receives notifications when new users are created in the database. When it happens, it creates a new user agent that will be the representation of the user in the system. The initial execution of the user data agent demands the creation of a user agent for each user already stored in the database;

User Agent: each user stored in the system has an agent that represents them in the system. This is the autonomous behavior, agents performing actions that the users should do. The user agent was designed in such a way that it can dynamically incorporate new roles. Each agent role perform specific actions according to the role that the user plays in the conference, such as chair, coordinator and author. An example of autonomous behavior is when the paper submission deadline expires and the user agent in the chair role will automatically distribute the papers to the committee members. Besides this example, most of the user agents are responsible: (i) for analyzing and discovering pending tasks for user agents based on the roles the users play in the system; and (ii) for asking the notifier agent to send email or SMS notifications;

Deadline Agent: this agent is responsible for monitoring the conference deadlines. This monitoring serves two purposes: (i) to notify the user agents when a deadline is nearly expiring; and (ii) to notify the user agents when a deadline has already expired;

Task Agent: this agent is responsible for managing the user tasks. It receives requests for creating, removing and setting the execution date of tasks. The requests are made by the user agents;

Notifier Agent: this agent receives requests from other agents to send messages to the system users. In the current implementation, it sends these messages through email and SMS.

3.4 Evolving the EC MAS-PL Architecture

In versions 2 and 3 of the EC system, the MAS PL architecture was developed to provide the minimum impact when the new optional and alternative features must be added. In this way, different architectural and design decisions were accomplished to facilitate the creation of different configurations (products) of the MAS-PL.

In the EC Version 2 of our MAS-PL, we adopt traditional design patterns to implement its variabilities (variable features). First, as we mentioned before, the integration between the web system and the environment agent was implemented using the Observer design pattern [12]. The Observer pattern was used to allow the (un)plugability of the agency features and maintain the alternative to have all the agency feature as optional. Second, we used the Role Object pattern [3] to better modularize the implementation of each of our agents (Section 3.3). The Role Object pattern models context-specific views of an object as separate role objects which are dynamically attached to or are removed from the core object. This pattern was mainly used to provide a base implementation of the user agents whose behavior can be incremented by attaching new roles (such as chair, author, committee, reviewer) to be played by these agents.

Listing 1.1. XML Configuration File

```
<bean id="ExpertCommitteeConfig"
     class="br.puc.maspl.config.ExpertCommittee">
  <property name="optionalFeatures">
    <list>
      <value>edit_user</value>
    </list>
  </property>
  <property name="roles">
    <map>
      <entry key="CHAIR"><ref bean="Chair" /></entry>
      ...
      <entry key="AUTHOR"><ref bean="Author" /></entry>
    </map>
  </property>
  <property name="agents">
    <map>
      <entry key="deadlineAgent"><ref bean="DeadlineAgent" /></entry>
      <entry key="taskAgent"><ref bean="TaskAgent" /></entry>
      <entry key="notifierAgent"><ref bean="NotifierAgent" /></entry>
    </map>
  </property>
  <property name="services">
    <list>
      <ref bean="AuthorService" />
      ...
      <ref bean="TaskService" />
    </list>
  </property>
</bean>
<bean id="Chair" class="br.puc.maspl.config.Role">
  <property name="optionalFeatures">
    <list>
      <value>automatic_paper_distribution</value>
    </list>
  </property>
</bean>
```

The EC Version 2 provided an improved modularization of the MAS-PL optional and alternative features through the adoption of design patterns. However, we noticed the need to accomplish new adaptations in the MAS-PL architecture in order to facilitate the production of different configurations of our MAS-PL. These required adaptations were: (i) to split the agent plans in small units to address only specific MAS features, because in the Version 2, the plans were implemented incorporating different features; and (ii) to define a mechanism to provide an easy way to configure the different features, including fine-grained properties. The EC Version 3 incorporated the implementation of these adaptations by providing: (i) a feature-oriented modularization of the agent plans; and (ii) a Spring-based mechanism to configure the main MAS-PL components. Both implementation decisions enable the automatic instantiation of our MAS PL architecture using product derivation tools, such as: pure::variants [27] and GenArch [5,4].

The customization of the MAS PL components using the Spring framework was accomplished by specifying a configuration file that aggregates different options of configuration of the MAS-PL, such as: (i) different functional features of the conference management base system (edit_user, paper_distribution) and respective properties; (ii) the different agents and the respective plans; and (iii) the different agent roles and respective plans. These important elements of the system were modeled using the bean abstraction of the Spring framework, which offers a model to build applications as a collection of simple components (called beans) that can be connected or customized using dependency injection and aspect-oriented technologies. Spring container uses a XML configuration file to specify the dependency injection on application components. This file contains one or more bean definitions which typically specify: (i) the class that implements the bean, (ii) the bean properties and (iii) the respective bean dependencies. Listing 1.1 illustrates a fragment of our MAS-PL configuration file.

4 Discussions and Lessons Learned

In this section, we present and discuss some lessons learned from our experience of development and evolution of the EC MAS-PL. Our lessons learned are related to the following main points: variability types, aspect-oriented refactoring, and adaptation of SPL methodologies.

4.1 MAS-PL Variability Types

SPL architectures address the implementation of different types of variable features, such as optional, alternative and OR-features [7]. In our development experience, we have found that in a MAS-PL, the occurrence of variable features varies not only in term of its functional features, but it also depends and is structured based on the agency features that the MAS-PL needs to address. According to [26], these two types of variability are classified as external and internal respectively. Since one of the main aims of the implementation of SPL architectures is to improve the modularization and management of their features, in a MAS-PL is essential to consider the agency features and to evaluate how the existing technologies can help to address them.

In our exploratory study, we have identified three types of variability in agency feature, all of them are internal. We believe that these three types can be considered in most of the MAS-PL, because they are really useful to improve the variability management of the MAS-PL. Next we briefly describe these three types:

New Autonomous Behavior. We had to introduce agents into the architecture when we added autonomous behavior to the system. The *Task Management* feature, for example, implied in the addition of a new agent in the system with a set of associated behaviors, which can be present or not, depending on the product being derived;

New Role for an Agent. Each role played by the users in the EC system has a corresponding role in the user agent of the MAS-PL. However, not all roles are mandatory, such as the role *Reviewer*. Thus, roles must be modeled in a way that they can be easily (un)plugged from the core architecture of the MAS-PL;

New Behavior for an Agent or Role. Some optional features have an impact inside the agent or the role. They allow specifying agent internal variabilities by defining new behaviors of agents. The *Conference Suggestion* feature is an example of such autonomous optional feature. The user agent, or more specifically the author role, can perform it. When a paper is registered in a conference, the author agent role perceives it and sends suggestions of related conferences for the author who has registered his/her paper.

In a MAS, it is common to specify the autonomous behavior features by means of the collaboration of several and different agents. When developing the versions 2 and 3 from the EC MAS-PL, we observed that the modularization of many of the agency features involved to codify different pieces of code related to agents and respective roles along different classes and agents from the SPL architecture. Figure 2 illustrates two examples of this scattering - the *Task Management* and *Conference Suggestion* features. It presents a colored indication that shows the elements (classes, interfaces, methods) related to the implementation of these features. Thus, it becomes a fundamental challenge to modularize some of these crosscutting features in order to allow their (un)plugability.

4.2 AO Refactoring

Recent research work presents the benefits of adopting aspect-oriented programming (AOP) techniques to improve the modularization of features in SPL [1,10], framework based [21] or multi-agent systems [14] architectures. The increasing complexity of agent-based applications motivates the use of AOP. AOP has been proposed to allow a better modularization of crosscutting concerns, and as a consequence to improve the reusability and maintenance of systems [19]. Among the problems of crosscutting variable features, we can enumerate: (i) tangled code - the code of variable features is tangled with the base code (core architecture) of a SPL; (ii) spread code - the code of variable features is spread over several classes; and (iii) replicated code the code of variable features is replicated over many places. All these problems can cause difficulties regarding the management, maintenance and reuse of variable features in SPL.

In order to promote improved separation of concerns, some crosscutting features that present the problems mentioned above are natural candidates to be designed and implemented using AOP. In our MAS-PL exploratory case study, we have found the following interesting situations to adopt AOP techniques:

(i) *modularization of the glue-code that integrates the web-based system (base code) with the agent features (new variable agency features)* - in our current implementation, this is addressed by the Observer design pattern [17] that is used to observe/intercept the execution of business methods of the services of the Business layer. AOP can be used to modularize the intercepted code that allows the agents monitor the execution of the web-based system. It facilitates the (un)plug of the agency features in the system. In our case study, 17 methods distributed among the services are intercepted to collect information for the agents; and
(ii) *modularization of the agent roles* - in the EC case study, we have used the Role OO design pattern to modularize the agent roles. We have noticed that the use of this pattern cannot provide an improved isolation of the agent role features, which is essential to SPL variability management. The implementation of the agent classes (e.g. `UserAgentCore` class) requires, for example, the activation and deactivation of the agent roles over different points of the execution of the agent behavior, such as agent initialization, execution of specific plans, etc. The adoption of AOP to modularize agent roles [13] is thus a better option to improve the modularization and evolution of the agent roles features.

We have recently refactored the EC MAS-PL version 3 in order to modularize the crosscutting features mentioned above using aspect-oriented programming techniques. This activity was part of an empirical quantitative study developed in order to compare the OO and AO implementations of the EC MAS-PL. Both implementations were compared in terms of size, separation of concerns and feature interaction metrics. The preliminary results from this quantitative study have shown that the AO implementation of the EC MAS-PL exhibits a better separation of concerns/features and reduced values for interactions between concerns, but on the other hand, it caused the increasing of the number of classes/aspects, operations and lines of code. For further details about this quantitative study, please refer to [22].

4.3 Adaptation of SPL Methodologies

Over the last years, many SPL methodologies have been proposed [26,15,2]. They cover a great variety of SPL development activities [26,6,15], related to domain and application engineering, as well to management processes. Some of these methodologies incorporate concepts and techniques from the object-oriented or component-based paradigms. Most of these SPL methodologies provide useful notations to model the agency features [23]. However, none of them completely covers their specification. Agent technology provides particular characteristics that need to be considered in order to take advantage of this paradigm.

Pena et al [25] identify current challenges to integrate the MAS and SPL software engineering approaches, such as: (i) management of evolving systems; (ii) need to provide new adapted techniques to cover distributed systems and the fact that agent-oriented software engineering does not cover typical activities of SPL development. There is some recent research work that addresses initial proposals to define a MAS-PL development methodology [24,9]. These proposals consider MAS methodology as a base and adapt it to document features of a product line. The main problems that we have observed in these approaches were [23]: (i) they do not offer a complete solution to address the modeling of agency features in domain analysis and design; and (ii) they suggest the introduction of complex and heavyweight notations that are difficult to understand when adopted in combination with existing notations (e.g. UML) and do not capture explicitly the separated modeling of agency features.

In our work, we have developed and evolved a web-based system by introducing the implementation of new variable agency features on its original architecture. We focused mainly on the use of OO techniques to modularize the implementation of the new agency features. The feature model was used to organize the SPL variabilities and guide us during the maintenance and refactoring of the different EC versions. The idea to introduce agency features in a web system was motivated by the growing need of this kind of systems to incorporate recommendations and alerts of pending tasks to their users. Based on the results of this exploratory study, we are currently investigating the need of proposing new extensions to existing SPL methodologies in order to model and modularize each of different agency features specified. The main aim is to allow an explicit documentation and tracing of these agency variabilities along the SPL development process. In particular, we are focusing on these two research directions: (i) documentation of MAS-PL architectures considering the integration of SPL and MAS proposed methodologies; and (ii) definition of a MAS-PL agile methodology to model their requirements and features. Preliminary results obtained to address these topics can be found in [23].

Our case study has defined an architectural style to increment web-based systems with new agency features. It allowed introducing new agency features related to recommendations and alerts to the system users. Since many web-based system are typically structured following the guidelines of the Layer architectural pattern, we are currently exploring the application of this same architectural style to different web-based systems in order to validate its applicability.

5 Conclusions and Future Work

This paper presented an exploratory study of development and evolution of a MAS-PL. We initially developed a traditional web-based system to support the process of conference management. After that, we evolved this system to incorporate a series of new agency features, which addresses autonomous behavior associated with recommendations to the system users. Different user agents and roles were implemented to modularize these features. The feature model was also adopted to drive the incorporation of the new features. As a result of our study, we presented a SPL architecture that allows to integrating agency features in traditional web-based system. Additionally, we presented a set of lessons learned from our study, related to: (i) the variability types encountered in our MAS-PL; (ii) the possibility of using aspect-oriented techniques to improve the modularization of agency features in our architecture; and (ii) the need of adaptation of existing SPL methodologies to allow the modeling of the different agency features during the SPL development.

We are currently extending the research work presented in this paper in several directions. We are working to define a base and lightweight SPL methodology that allows the specification and documentation of autonomous behavior along the domain and application engineering processes. We are also interested to explore the use of our MAS-PL architecture to incorporate autonomous behavior in other web-based systems in order to validate it as an architectural style [28]. Finally, we are also investigating what is the impact of OO and AO implementations of a MAS-PL to the automatic product derivation supported by existing tools [5,27].

References

1. Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., Lucena, C.: Refactoring product lines. In: GPCE 2006, pp. 201–210. ACM, New York (2006)
2. Atkinson, C., Bayer, J., Muthig, D.: Component-based product line development: The kobrA approach. In: Donohoe, P. (ed.) SPLC 2000, pp. 289–309 (2000)
3. Bäumer, D., Riehle, D., Siberski, W., Wulf, M.: The Role Object Pattern. In: PLoP 1997 (1997) (submitted), `citeseer.ist.psu.edu/baumer97role.html`
4. Cirilo, E., Kulesza, U., Coelho, R., Lucena, C., von Staa, A.: Integrating Component and Product Lines Technologies. In: Mei, H. (ed.) ICSR 2008. LNCS, vol. 5030, pp. 130–141. Springer, Heidelberg (2008)
5. Cirilo, E., Kulesza, U., Lucena, C.: A Product Derivation Tool Base on Model-Driven Techniques and Annotations. Journal of Universal Computer Science (2008)
6. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston (2002)
7. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Longman, Amsterdam (2000)
8. Deelstra, S., Sinnema, M., Bosch, J.: Product derivation in software product families: a case study. Journal of Systems and Software 74(2), 173–194 (2005)
9. Dehlinger, J., Lutz, R.R.: A Product-Line Requirements Approach to Safe Reuse in Multi-Agent Systems. In: SELMAS 2005, pp. 1–7. ACM Press, New York (2005)
10. Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F., Dantas, F.: Evolving software product lines with aspects: An empirical study on design stability. In: ICSE 2008, pp. 261–270. ACM, New York (2008)
11. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley Professional, Reading (2002)

12. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley, Reading (1995)
13. Garcia, A., Chavez, C., Kulesza, U., Lucena, C.: The role aspect pattern. In: EuroPLoP 2005, Isree, Germany (2005)
14. Garcia, A., Lucena, C., Cowan, D.: Agents in object-oriented software engineering. Software Practice Experience 34(5), 489–521 (2004)
15. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley Longman Publishing Co., Inc., Redwood City (2004)
16. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley and Sons, Chichester (2004)
17. Hannemann, J., Kiczales, G.: Design pattern implementation in Java and aspectJ. In: OOPSLA 2002, pp. 161–173. ACM, New York (2002)
18. Jennings, N.R.: An agent-based approach for building complex software systems. Commun. ACM 44(4), 35–41 (2001)
19. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
20. Krueger, C.W.: Easing the transition to software mass customization. In: PFE 2001, London, UK, pp. 282–293. Springer, London (2002)
21. Kulesza, U., Alves, V., Garcia, A.F., de Lucena, C.J.P., Borba, P.: Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming. In: Morisio, M. (ed.) ICSR 2006. LNCS, vol. 4039, pp. 231–245. Springer, Heidelberg (2006)
22. Nunes, C., Kulesza, U., SantÁnna, C., Nunes, I., Lucena, C.: On the modularity assessment of aspect-oriented multi-agent systems product lines: a quantitative study. In: SBCARS 2008, Porto Alegre, Brazil (2008)
23. Nunes, I., Kulesza, U., Nunes, C., Lucena, C.: Documenting and modeling multi-agent systems product lines. In: SEKE 2008, Redwood City, USA (2008)
24. Pena, J., Hinchey, M.G., Resinas, M., Sterritt, R., Rash, J.L.: Designing and managing evolving systems using a MAS product line approach. Science of Computer Programming 66(1), 71–86 (2007)
25. Pena, J., Hinchey, M.G., Ruiz-Cortés, A.: Multi-agent system product lines: challenges and benefits. Communications of the ACM 49(12), 82–84 (2006)
26. Pohl, K., Bóckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
27. Pure systems. pure-systems GmbH (2008), http://www.pure-systems.com/
28. Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Englewood Cliffs (1996)
29. Wooldridge, M., Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 1–28. Springer, Heidelberg (2001)

Combining JADE and Repast for the Complex Simulation of Enterprise Value-Adding Networks

Min-Jung Yoo and Rémy Glardon

EPFL-STI-IGM-LGPP
Swiss Federal Institute of Technology
Station 9, CH-1015 Lausanne, Switzerland
{min-jung.yoo,remy.glardon}@epfl.ch
http://lgpp.epfl.ch

Abstract. This paper describes how to combine the JADE agent platform with Repast-provided simulation functions for rapidly developing an environment for the simulation of complex agent model. The main motivation comes from our requirements concerning the simulation of enterprise Value-Adding Networks, whose ultimate objective is to analyse management performance. JADE is useful for creating and deploying a distributed agent organisation modelling enterprise workflow model and supply chain network, as well as for system monitoring at the level of agents and communications. Integrating dynamically human interaction during the simulation is relatively easy. Repast was developed as a general purpose framework for agent based simulation with appropriate graphic interfaces. Their combination makes it possible to construct complex model of multi-agent organisation whose execution states can be observed from users and the global simulation results can be used for performance analysis. Their extension and composition mechanisms are described using a case study of a manufacturing enterprise.

Keywords: Simulation, supply chain, agent platform, enterprise modelling, JADE, Repast.

1 Introduction

The continuous trend towards global operation leads enterprises to create strong links with a wide-ranging, complex network of partners. These networks, called Value-Adding Networks (VAN), constitute highly complex systems that operate in very dynamic and unpredictable environments, which render intuitive decisions, based on experience and common sense, impossible or extremely risky.

Modern enterprises therefore require an adequate support tool which helps them in decision making processes (such as where to manufacture, assemble, and store what for which market) by making it possible to test several alternative supply chain configurations. Simulation-based approaches are often chosen as one of the best solutions for such tools thanks to their predictable features [1][2][3][4]. Our ultimate goal is to develop the model of enterprise VAN for different supply chain configurations and to test it varying performance parameters so as to analyse different results from simulation. In that context, rather than developing all the functions from scratch, we are interested in using existing toolkits as much as possible.

There are already many simulation frameworks and agent-based toolkits which are available for simulation approaches (Section 2). For the purpose of achieving our objectives, we have used JADE (Java Agent DEvelopment Framework [17]) combining it with supplementary functions from Repast (REcursive Porous Agent Simulation Toolkit [18]) such as step-by-step simulation and developing statistical monitoring functions.

Even though such an extension was achieved for our particular needs, the resulting outcome is sufficiently general to be applied to other contexts. The contribution of this paper is therefore to detail how the combination of JADE and Repast was achieved so that other research can benefit from the result.

This paper is organized as follows. Section 2 discusses the context of enterprise VAN simulation and our motivation in combining two platforms as a basic environment for that purpose. The fusion of the platforms, which were necessary to enable their integration into a common development environment, are detailed in Section 3. A case study example using the extended two-agent platforms is given in Section 4. Finally, conclusions are presented in Section 5.

2 Simulation Approach: Context, Motivation and Objective

This section discusses what is meant by enterprise VAN modelling, why JADE was used and why the decision was made to combine it with Repast.

2.1 Context: Enterprise Simulation for Supply Chain Performance Analysis

Enterprise VAN modelling and simulation involves building a functional model of enterprise business processes and value adding activities in production including external supply chain actors, and runing the model during a pre-determined time span. This approach is described below.

1. All of the involved business departments, such as production and purchase planning, manufacturing, or inventory management have to be modelled, including their workflow or enterprise-specific management and planning procedures.
2. For the purpose of testing the external impacts of their Supply Chain, external actors such as Distribution Logistics, Suppliers, or Clients should be modelled according to their behavioural characteristics.
3. The simulation model must integrate not only key flows of materials or information (contracts and order forms) but also financial flows such as payments or costs, which can be influenced by variations in currency conversion rates or the location of transactions.

4. For the purpose of improving the clarity of the simulation for users, the changing states of modelled elements should be visually monitored. This might include stock levels which changes dynamically according to input-output transactions, or the global job load of manufacturing centers.
5. The simulation results, which show the detailed transaction information in inventory or production operation records, should be retrieved afterwards for the purpose of performance analysis.

Another important aspect to integrate into the simulation is the possibility of user interactions during run-time. This means that the tool should be able to respond interactively to user inputs, such as a client order for a huge amount of a product, or output transactions from stock, so that the simulation shows the real impact of such changes on the whole organisation.

Such an approach is of great interest to supply chain management and decision support. Within the design phases of a supply network or of production nodes, it is possible to compare the simulation results of different variations, or to evaluate the effects of changes in a planning scheme or management policy. In addition, it can help to optimize the processes of value-adding activities, including sales planning, inventory planning, and distribution. Whereas other simulation approaches ([3] [4]) partially achieved some selected objectives using an abstract and simplified enterprise model, our work aims at achieving a complex simulation with the help of an enterprise model closely reflecting the real world functional details of the enterprise.

2.2 Motivation: JADE as a Platform for Enterprise VAN Modelling and Simulation

The above discussion reveals not only the complexity of the modelled system but also the necessity and difficulty of developing a modelling framework that can support the creation of such complex models. There are many simulation packages either commercially available today, such as AnyLogic [20], Arena [19], or open sources (see Section 2.3), to mention only a few of them. They provide good modelling elements and various types of histograms developing functions and statistical analysis. Nevertheless, they were not conceived to satisfy all the needs related to the above-mentioned model characteristics. Most of them are based on waiting queues; even though they are able to model material flow, albeit with limited flexibility, they are not capable of satisfactorily integrating either the information or the enterprise management procedures [3]. Integrating user interaction during run-time is not straight-forward with such tools. In order to develop such a complex modelling framework, the use of a multi-agent architecture is strongly recommended [5] [12] [16].

From our point of view, the characteristics of our simulation model are close to the decentralised organisation of autonomous agents in an "open and dynamic environment" [9] which as studies in [10] [16]. In the context of our former research experiences, JADE was experimented and received a good evalusation for that purpose. JADE provides the following characteristics.

Generic agent structure. The generic agent structure provided by JADE helps to encapsulate different levels of business activities within an autonomous software agent, separating them from other concerns such as agent communication and collaboration.

ACL message semantics. FIPA specifies the ACL (Agent Communication Language) as a means of interaction among agents. Even though the language suffers from its limited semantic richness [7], ACL semantics ("performatives") are mainly based on human language act theory. By using a unified meaning of expression in communication, it is possible to model seamless communication for agent-to-agent and agent-to-human interaction. It also provides the possibility of flexible modification following agent acquaintance.

Dummy Agent. Among several utilities offered by JADE, "Dummy Agent", a user interfacing agent, is useful for the integration of human user interaction.

Agent organisation modelling. The platform provides several utilities for managing agent collaboration in an open organisation, which consists of communication lines, an agent directory facilitator, and the notion of agent services.

Several evaluation reports on performance tests concerning agent development tools already concluded that JADE is highly efficient in terms of its performance on agent message transport layer [13], internal database access and message exchange capabilities [14], or its appropriateness as an efficient tool for a manufacturing control application [16] . These are the reasons why JADE was chosen for the modelling platform.

The main issue is that JADE does not provide simulation capabilities, such as synchnonising agents with the step-by-step simulation cycle or tools for statiscal analysis concerning the changes of some internal agent states, which were well integrated in other simulation tools [6]. Rather than developing these missing points from scratch, our decision was to study first of all a possible means of combining JADE with another toolkit for the purpose of rapid development.

2.3 Objective: Incrementing JADE Features towards a Repast-Enabled Simulation Environment

In [6], several software platforms, Swarm[21], NetLogo[22], and Repast, were reviewed as scientific agent-based models (ABMs) by comparing these platforms based on several criteria such as model structure, scheduling, or execution speed, and other general agent development issues. According to the authors, Repast is a good JAVA platform simulation implementing almost all of Swarm's functions with other added capabilites, such as the ability to reset and restart models from the graphical interface while at the same time guaranteeing good performance on execution speed.

Repast received a less favourable evaluation concerning its modelling structure in comparison with other candidates. In the absence of a platform-provided agent structure (at least with the version of Repast 3), any user-defined Java class can be defined as an "agent" in that environment. However, ironically, this weakness becomes a positive characteristic in order to be combined with JADE because the agent model itself in the simulation must be based on the JADE proposed agent framework. As a result, we decided to study the feasibility of using Repast in order to complement the JADE-based supply chain modelling. More particularly, the following aspects are interesting to be integrated with the JADE-based enterprise model.

Simulation scheduling and control. Repast, along with other platforms in the same family, is particularly developed as a general-purpose platform for step-by-step simulation for which a rich set of functions is predefined. One of these is the generic simulation model (SimModelImpl) which is provided with some predefined template methods for making models and planning schedules. In particular, the "buildSchedule" method involves activating all agents in the simulation model.

Visualisation. Repast also supports efficient display functions used to show the changing state of observable data according to some predefined performance criteria. Such functions are useful for rapidly developing monitoring functions, illustrating simulation progression in real-time.

Other features. Other functions, such as controlling the total number of simulations, or saving the graphs, are also helpful for simulation monitoring, data saving and retrieving.

3 JADE and Repast Combination

For the purpose of efficiently using different characteristics within a unified agent development context, the following modifications and improvements were achieved.

1. Creation of a specialized JADE agent class and Simulation Cycle Synchronisation behaviour.
2. Creation of a specialised synchronisation behaviour.
3. Modification of the JADE ACL message scheme to include temporal aspects.
4. Implementation of a specialised Jade Behaviour class in order to use Repast-provided display functions: named "ObservableBehaviour" hereafter.
5. Capacility of recognizing current time according to the simulated Gregorian calendar

3.1 SCAgent

SCAgent class is a subclass of the JADE "Agent" class with the following extensions:

1. The possibility of representing the notion of "simulation time" in accordance with the agent lifecycle. For instance, if one simulation cycle is equivalent to one-day length of time of the simulated environment, then the agent activities of a day must be terminated within one simulation cycle, which corresponds to a clock tick of Repast.
2. The ability to handle a special type of agent message ("TickerACLMessage", see Section 3.2).
3. The ability to track global simulation time ("currentTick" value) and corresponding date notion according to Gregorian calendar. This is necessary in order to synchronise with other agents and to control the simulation in terms of current simulation step value.
4. Integration of "Simulation Cycle Synchronisation" Behaviour which has to be activated at the end of the SCAgent's one-tick cycle (see below).

The notion of simulation time discussed in the first point means a synchronised value with the clock tick (one step of simulation) in Repast. It can be used in association with notions such as "what day is today?" (global time of the simulated environment), and "how long does it take?" (duration) of certain manufacturing operations.

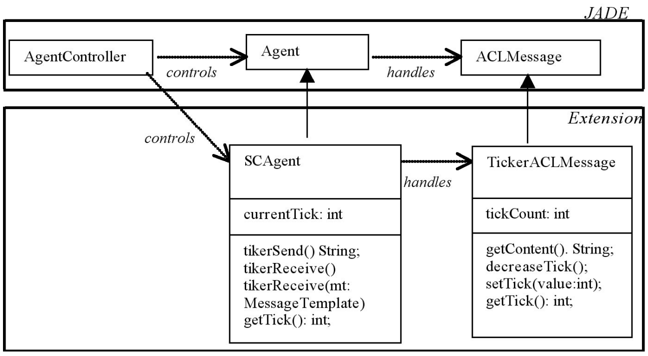


Fig. 1. SCAgentController, SCAgent, and TickerACLMessage: subclass relationships and associations

Simulation Cycle Synchronisation Behaviour. Being based on the first version of this prototype presented at AOSE 2008, the JADE-Repast fusion have improved, and this behaviour simplifies the handling of simulation cycle synchronisation within the subclasses of SCAgent. The Simulation Cycle Synchronisation Behaviour simply increments the tick value and make the agent suspend itself.

3.2 TickerACLMessage and Associated Handling Mechanisms

This is a subclass of JADE's "ACLMessage", integrating the notion of simulation time (Fig. 1). It represents the duration of the number of simulation steps which is needed for the delivery of messages from a sender agent to a receiver. This value must be initialised at its creation and decreased while the simulation continues. While a normal ACLMessage can be opened and verified by an agent as soon as the message is delivered to the receiver agent's message box, a TickerACLMessage must not be opened until the tick value reaches zero. This test is achieved by an SCAgent ("tickerReceive" method).

This modification makes it possible to model temporal aspects related to the material and information flows. The importance of integrating the temporal aspect will be further discussed in Section 4.2.

3.3 ObservableBehaviour

As was discussed above, Repast already provides special monitoring functions for users, ranging from the state of individuals to statistics of the population. These kinds of observation or display functions should be implemented using special interfaces - "DataSource" and "Sequence" (see Repast programming guide on the web site [18] for more details). For the purpose of providing the same type of observability, a specialised JADE "Behaviour" class was created to implement these "interfaces" (Fig. 2). If the execution of an agent behaviour concerns state changes, such as stock evolution, or the progress of manufacturing processes, statistical data that is to be observed can be exported to the appropriate graphic user interfaces in Repast by this method (c on Fig. 5).

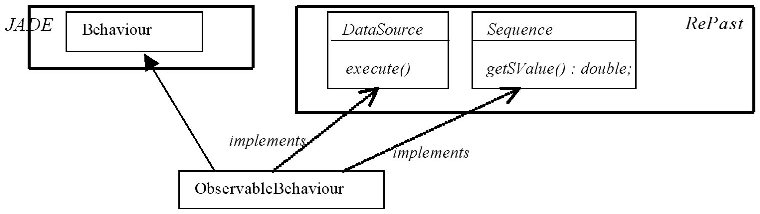


Fig. 2. ObservableBehaviour, a subclass of JADE's "Behaviour" class

3.4 Synchronisation between Simulation Cycle and Gregorian Calendar Type Value

The improved version of the SC agent class integrates the notion of the Gregorian Calendar which is initialised with a given date valule (for example, 1st January 2007 at the start of first cycle). This change makes it possible to know what day is today (work day or week-end). By associating the working calendar specific to each department, the simulation results show the impact of changing the work day, or taking into account seasonal characteristic analysis.

3.5 Overall Configuration

Fig. 3 shows the overall configuration of the composed architecture. The simulation step control from Repast arrives at SCAgents in the JADE environment by passing through the corresponding AgentControllers. After activation by its controller ("resume" method), an SCAgent accomplishes its activities, which must be done within a simulation cycle, and then suspends itself, while waiting for the next activation from its AgentController. If an agent includes an ObservableBehaviour object, changes in the internal agent state can be observed from the Repast monitoring environment (see also Fig. 5 and Section 4.3). Regardless of the type of internal behaviour objects, every agent on the composed platform is able to communicate using either standard ACL messages or extended TickerACL messages.

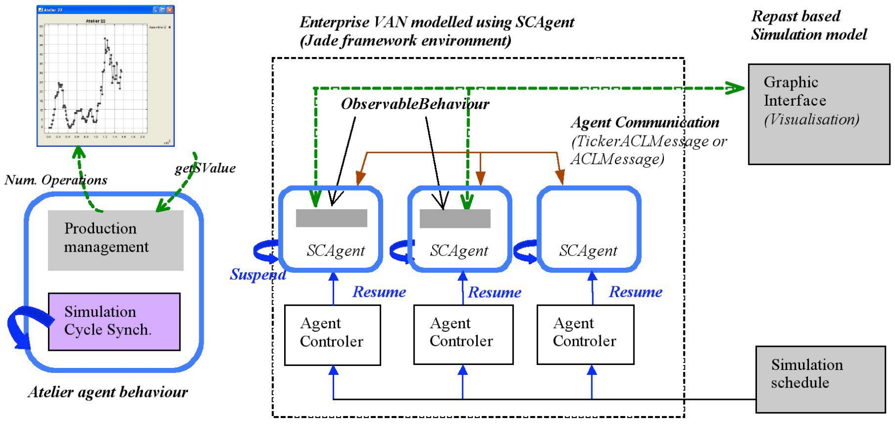


Fig. 3. Overall configuration of the composed architecture of JADE-Repast

4 Application to a Case Study of a Swiss Manufacturing Firm

Currently, the modelling environment is being applied to a real-world case study concerning a Swiss manufacturing company with a view to analysing current and another possible Supply Chain configurations.

4.1 Model Description

The enterprise VAN model is composed of the following supply chain actors (Fig. 4).

1. A set of agents representing the enterprise (business departments or actors), such as “Supply Chain (SC) Manager”, “Stock Manager”, “Purchase Manager”, or assembly and manufacturing centre (“Atelier” on the figure) , in order to design activities and business processes in detail.
2. Agents representing other supply chain actors: suppliers and transport logistics.

Each agent is modelled as a subclass of SCAgent implementing necessary value-adding activities by using one of the subclasses of Behaviour or newly added ObservableBehaviour objects. For example, the following text describes the agent behaviour of “Stock Manager” including the default “Simulation Cycle Synchronisation” and several other behaviours.

The Stock Manager receives manufacturing orders from the SC Manager. If there are sufficient materials to produce the requested final product, Stock Manager takes them and send them, together with a Manufacturing Order to the appropriate ateliers (“Kitting and Launching” - CyclicBehaviour). In order to find pertinent ateliers, the Stock Manager refers to the internal database of the “Manufacturing Operation Series” of production. If there are not enough materials, then the Manufacturing Order will be kept until the materials arrive at stock. Inventory Management keeps track of input and output stock transactions (“Inventory Management” - ObservableBehaviour).

The Stock Manager manages the bill of materials of these products. The production strategy is based on a pull mechanism, that is to say, the enterprise produces only the quantity ordered by its clients.

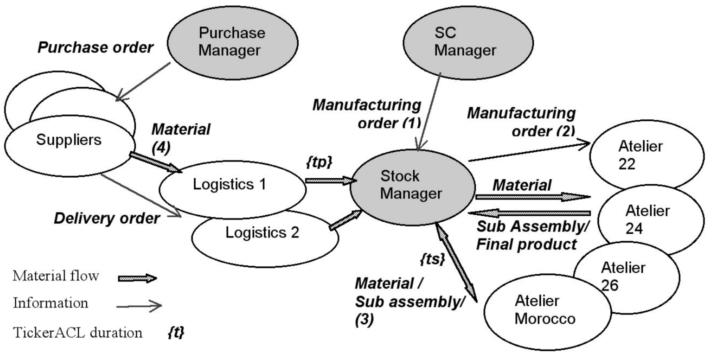


Fig. 4. The simplified supply chain organisation of the case study

4.2 Communication Model for Material and Information Flows

Apart from agents, the modelling environment already provides some predefined documents and primitive data types in order to model the material and information flows, such as “Order of Product”, “Bill of Materials”, “Manufacturing Order”, “Purchase Order”, “Ship”, “Product Delivery”, “Operation” and many more.

Different agents communicate in order to share information and materials according to their value-adding activities. Here are some examples:

1. SC Manager sends a “Request” message of “Manufacturing Order” to the Stock Manager ((1) on Fig. 4).
2. Stock Manager sends an “Inform” message of “Manufacturing Order” to the Ateliers ((2) on Fig. 4.).
3. The Ateliers send their final products and associated documents (“Ship”) to Stock Manager during a predefined delay time ts ((3) on Fig. 4).
4. The Suppliers deliver materials to Transport Logistics and then to Stock Manager (“Product Delivery”) during the delay time tp ((4) on Fig. 4).

In JADE, when an agent sends an ACL message, it is immediately delivered to the receiver agent. Meanwhile, in a real-world example, the document-sending of “Manufacturing Order” can take place within a day, while the delivery of products depends on the logistical means and locations. For example, the product shipped from Morocco requires two to three business days whereas it may be dealt with immediately from other centres (Atelier 22, 24, 26 in Switzerland). That is the reason why they were modelled using different types of agent communication mechanisms, either standard ACL Communication or modified TickerACL Communication. Here we can see the importance of using the TickerACLMessage, which enables us to model this difference in delivery duration for different flows.

4.3 SCAgent Class Creation with an “Observable Behaviour” Object

The manufacturing centres (atelier) are modelled as a subclass of SCAgent including a specific behaviour class, “Production Management”, which is a kind of “ObservableBehaviour” (left-hand side on Fig. 3).

ProductionManagement is a subclass of “CyclicBehaviour” (JADE) which implements Repast’s interfaces “DataSource” and “Sequence”. When an Atelier agent receives a Manufacturing Order, the agent saves the corresponding operations process within its job queue. Each Operation object keeps track of the number of steps required for the purpose of finishing the manufacturing operation.

Production process execution is simulated by decreasing the number of steps associated with every Operation object. Each time the Repast’s display function activates the “getSValue” method of “Sequence” interface, ProductionManagement returns the total number of operations that the agents are currently dealing with.

4.4 Simulation

The agent model was successfully implemented and run on the composed agent execution environment (Fig. 5). The following enterprise data was integrated as simulation-enabling data.

- Product Bill of Materials concerning selected top ten final products
- Manufacturing Operation series (Bonding, reception, packaging, etc.) concerning these products
- Launched Manufacturing Order in 2007
- Purchase Order history in 2007
- Client Orders in 2007

The simulation scenario corresponded to the annual client order data, purchase order history and launched production orders. The enterprise historical databases were stored in Microsoft's Excel spreadsheets and retrieved with the help of Java Excel APIs [23] for the purpose of initialising agents with corresponding data (e.g., SC Manager with Client orders, Purchase Manager with Purchase order history). The results in Fig. 5 shows an intermediate status on about 7th June in the middle of one year simulation from January to December 2007.

During the simulation, the material and information flows were visualised in the Sniffer Agent's graphic window as agent message passing (b). In the middle of the simulation, users could easily suspend it by using the button (a) and

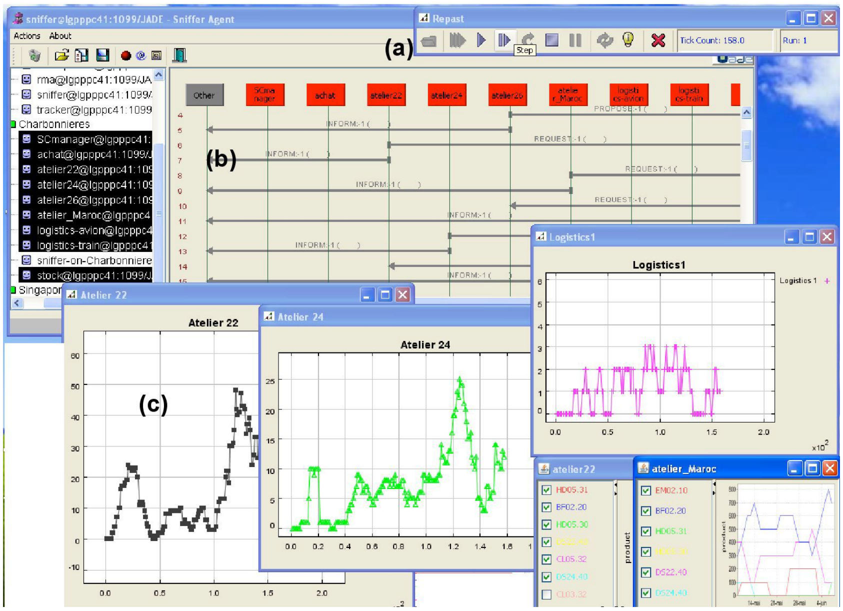


Fig. 5. Screen Copy: during the simulation of the case study

send an arbitrary product order with the help of the Dummy Agent interface to observe the impact of this supplementary order on the graphic interfaces.

5 Conclusion

This paper describes how to combine JADE with Repast-provided simulation facilities. The JADE framework is useful for deploying collaborative agents in an open environment as well as for monitoring the organisation. The extended agent communication mechanism is appropriate for flexible modelling of the supply chain material and information flows. During the simulation, the Repast framework controls the simulation steps and makes it possible to show changes in agent states which result from the agent activities and communications taking place on the other side of the agent platform - JADE. The combined use of two platforms is possible thanks to some extensions discussed in this paper thereby enabling savings to be made in development costs and time.

Lessons learned. The most important lesson was that we could prove the suitability of the approach. Using two distinctive platforms within a unified environment was a new idea. The positive result shown in this paper opens up a new potential for "open-source" users and the opportunity to create other interesting combinations.

Another lesson was that FIPA-compliant ACL semantics seem quite promising in terms of modelling supply chain flow and user interaction, as well as later exploration of the agent organisation in the real context of supply chain management, or in an open organisation such as in [10]. We should also mention that the agent organisation mechanism in JADE, for instance the notion of "Agent Service Description", was useful for describing the different roles of SC actors and their relationships.

Benefits from using JADE. The benefits from using JADE are therefore the modelling power provided by JADE for developing such systems. The supply chain model of extended enterprises is a distributed organization of autonomous business actors. These include the notions of workflow management, collaboration and information sharing among actors, cooperation protocols, and so on. Using JADE makes it easy to develop such an organization based on an individual model close to the real agent behaviour.

Another important and interesting point is that Jade Web Services Integration Gateway [8] makes agents possible to access to Web services. It enables agents to use numerous Web services during the simulation. For example, there are several currency converters available on standard Web service. By directly using this function during the simulation of a payment operation , the financial flows can be correctly modelld so that cost performance analysis becomes more credible.

Benefits from using Repast. As was described before, the Repast-provided GUIs help developers to rapidly construct simulation prototypes. A pre-defined agent controlling mechanism makes it easy to integrate each JADE agent behaviour and synchronise it with the environmental simulation clock. If the Repast part should not be provided, then they would have to be developed from scratch. Ultimately, it is up to the developers whether to use them or not. We preferred to benefit from the existing simulation libraries. We are planning to test other features from Repast such as file output function (for the purpose of facilitating data recording actions) and geographical data integration, or the integration with the version of Repast Simphony.

Future work. One of our current studies concerns the design of a high-level abstract agent architecture in order to further generalise the SCAgent structure. We are continuing to develop a methodological approach to SC sensitivity analysis and the test of robustness for the enterprise model presented in this paper. The integration of the financial flows by accessing directly to a Web Service is ongoing.

Given that we are aiming to use the enterprise simulation for a broad range of decision support and supply chain management, a consideration of other related issues is also very important. Our project is therefore also exploring the possibility of integrating the enterprise competence model [11] using an ontology language, or considering environmental aspects and ecological issues, which could be an important future factor in the strategic decisions of supply chain modelling.

The system extension on JADE and Repast presented here can be used for other types of simulation models if the requirements from the simulation environment are identical to which descirbed in Section 2. We are planning to open the API, which concerns the extension of JADE-Repast discussed in this paper, to publicly available resources in the near future.

Acknowledgments. This work is supported by the Innovation Promotion Agency of Switzerland (CTI grant 9160.1) and is currently ongoing. The authors thank all industrial partners in this project and Mr. S. Naciri from LGPP-EPFL for their close collaboration. The authors thank Dr. F. Bellifemine from Telecom Italia for his helpful comments on JADE extension. Our heartfelt thanks go as well to anonymous reviewers whose useful comments helped us to improve the paper.

References

1. Terzi, S., Cavalieri, S.: Simulation in the supply chain context- a Survey. Computers in Industry 53, 3–16 (2004)
2. Robinson, S.: General concepts of quality for discrete-event simulation. European Journal of Operational Research 138, 103–117 (2002)
3. Ridall, C.E., Bennet, S., Tipi, N.S.: Modeling the dynamics of supply chains. International Journal of Systems Science 31, 969 (2000)
4. Van der Zee, D.J., Van der Vorst, J.G.A.J.: A modeling Framework for Supply Chain Simulation - Opportunities for Improved Decision Making. Decision Sciences 36, 65 (2005)

5. Parunak, H.V.D., Savit, R., Riolo, R.L.: Agent-Based Modeling vs. Equation-Based Modeling- A Case Study and Users' Guide. In: Sichman, J.S., Conte, R., Gilbert, N. (eds.) MABS 1998. LNCS (LNAI), vol. 1534, pp. 10–25. Springer, Heidelberg (1998)
6. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based Simulation Platforms: Review and Development Recommendations. Simulation 82(9), 609–623 (2006)
7. Petrie, C., Bussler, C.: Service Agents and Virtual Enterprises - A Survey. IEEE Internet Computing 7(4), 68–78 (2003)
8. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology (2007)
9. Luck, M., Ashiri, R., D'Inverno, M.: Agent-Based Software Development. Artech House (2004)
10. Nguyen, D., Thompson, S.G., Patel, J., Teacy, L.W.T., Jennings, N.R., Luck, M., Dang, V., Chalmers, S., Oren, N., Norman, T.J., Preece, A., Gray, P.M.D., Shercliff, G., Stockreisser, P.J., Shao, J., Gray, W.A., Fiddian, N.J.: Delivering services by building and running virtual organisations. BT Technology Journal 25(1) (2006)
11. Yoo, M.-J., Furbringer, J.-M., Naciri, S., Glardon, R.: Integrating Competence Model and the Multiagent Simulation of Value-Adding Networks. In: Conférence internationale de Modélisation et Simulation, Paris, France (2008)
12. Muller, J.-P.: Towards a formal semantics of event-based multi-agent simulations. In: International workshop on Multi-Agent Based Simulation, MABS Estoril, Portugal (2008)
13. Shakshuki, E., Jun, Y.: Multi-agent Development Toolkits: An Evaluation. In: Orchard, B., Yang, C., Ali, M. (eds.) IEA/AIE 2004. LNCS (LNAI), vol. 3029, pp. 209–218. Springer, Heidelberg (2004)
14. Chminel, K., Tomiak, D., Gawinecki, M., Kaczmarek, P., Szymczak, M., Paprzycki., M.: Testing the Efficiency of JADE Agent Platform. In: Third International Symposium on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (2004)
15. Vrba, P.: JAVA-Based Agent Platform Evaluation. In: Mařík, V., McFarlane, D.C., Valckenaers, P. (eds.) HoloMAS 2003. LNCS (LNAI), vol. 2744, pp. 47–58. Springer, Heidelberg (2003)
16. Vrba, P.: Simulation in agent-based control systems: MAST case study. International Journal of Manufacturing Technology and Management 8(1-3), 175–187 (2006)
17. JADE, `http://JADE.tilab.com`
18. Repast, `http://repast.sourceforge.net/repast_3/index.html`
19. Arena, `http://www.arenasimulation.com`
20. AnyLogic, `http://www.xjtek.com`
21. Swarm, `http://www.swarm.org`
22. NetLogo, `http://ccl.northwestern.edu/netlogo`
23. Java Excel API, `http://jexcelapi.sourceforge.net`

OperA and Brahms: A Symphony?
Integrating Organizational and Emergent Views on Agent-Based Modeling

Bart-Jan van Putten[1,2], Virginia Dignum[2],
Maarten Sierhuis[2,4], and Shawn R. Wolfe[3]

[1] Utrecht University, Intelligent Systems Group,
PO Box 80089, 3508 TB Utrecht, The Netherlands
{bartjan,virginia}@cs.uu.nl
[2] RIACS / NASA Ames Research Center,
Mail Stop B269-1, Moffett Field, CA 94035, USA
Maarten.Sierhuis-1@nasa.gov
[3] NASA Ames Research Center, Intelligent Systems Division,
Mail Stop B269-2, Moffett Field, CA 94035, USA
Shawn.R.Wolfe@nasa.gov
[4] Delft University of Technology, Man-Machine Interaction Group,
Mekelweg 4, 2628 CD Delft, The Netherlands
M.Sierhuis@tudelft.nl

Abstract. The *organizational view* on work systems focuses on the desired outcomes of work (i.e., the work process) while the *emergent view* focuses on how the work actually gets done (i.e., the work practice). Often a gap exists between these two, because workers pursue individual objectives in addition to the organizational objectives. Agent-based modeling and simulation can be used to improve work systems and thereby organizational performance. Current modeling and simulation frameworks only represent either one of the two views. In order to model both views, we propose an integration of two modeling and simulation frameworks, OperA and Brahms. Using the integrated model, we are able to run simulations that show to what degree work practice differs from work processes.

1 Introduction

Organizations are intentionally formed to accomplish a set of common objectives, defined by the policy makers of the organization. People that work for those organizations often only partially pursue the global objectives of the organization. Workers often pursue their individual objectives as well, frequently resulting in a gap between the a priori designed flows of tasks and procedures reflecting the ideal activity of the organization (i.e., the work process), and the activities that actually get things done (i.e., the work practice) [2]. This gap does not exist only because of the difference in objectives between individuals and the organization, but also because many policy makers abstract from work practice when they design work systems (i.e., business operations). For example, it is uncommon for a job description to include 'socialize with co-workers', 'drink coffee', or 'read e-mail'.

Human Resource Management research has recognized that ultimately employee behaviors, rather than management practices, are the key to value creation in organizations [3]. Policy makers of successful organizations therefore want to understand work practice and align it with the organizational objectives. Modeling and simulation can support the description, prescription, and prediction of work practice [15], but also need to show in what ways work practice deviates from the organizational objectives.

Agent-based modeling and simulation used to focus on either the individual, 'micro' level in a way that the collective behavior emerges from individual actions (i.e., the bottom-up, emergent view), or on the global objectives and desired collective behavior at a 'macro' level (i.e., the top-down, organizational view). In order to bridge the gap between what the policy makers of an organization want and what the people do, a Work Systems Modeling and Simulation (WSMS) framework is needed that integrates the organizational and emergent views. This will allow policy makers to analyze effects of the micro on the macro level and vice-versa [4].

Related approaches, such as S-Moise+ [11], RNS2 [21], and [18] are similar to our research, in the sense that all aim to develop organizational models to support different levels of coordination and autonomy. However, the difference is that they aim to develop open, heterogeneous multi-agent systems from an engineering perspective, whereas we aim to develop models of work practice from a human-centered perspective, where norms may be violated. Furthermore, our approach demonstrates that the model that is based on the organizational view can be combined with the model that is based on the emergent view. Any agent behavior is allowed, but monitored, and possibly sanctioned if organizational rules are violated.

In this paper we will show how two multi-agent modeling frameworks, OperA [5], a methodology developed to represent and analyze organizational systems, and Brahms [17], a language developed to describe and simulate work practice, have been integrated into one WSMS framework. By running simulations using the integrated model, it is possible to determine in what ways work processes differ from work practice. The results of these simulations are used by policy makers, for example to:

- change organizational policies (e.g., make them more realistic or more challenging), or
- implement new forms of control in the organization to enforce different behavior, or
- help individual workers in adopting new work behavior, e.g., with protocols that are known to conform to the organizational policies.

This paper is organized as follows: section 2 introduces the case of Collaborative (Air) Traffic Flow Management, section 3 describes OperA and Brahms, section 4 describes their integration into one framework, section 5 describes validation of the framework, section 6 describes related work, and finally section 7 concludes this paper.

2 Case Study: Simulation

Air traffic in the United States of America (USA) has been projected to increase as much as threefold by the year 2025. A simulation of this level of traffic with the current air traffic systems shows a disproportionate and unacceptable increase in average delay per flight. As a result, NASA is researching new technologies and approaches to handle the problems associated with this projected traffic increase. One promising area is Collaborative Traffic Flow Management (CTFM), which seeks to increase the amount of collaboration between the controllers of the airspace (i.e., the Federal Aviation Administration (FAA)) and the many airlines that use the airspace to find beneficial solutions to traffic flow problems. A *concept of operations* (i.e., a future work process) has been suggested as a specific way to address the problem [14]. Figure 1 is an example of a work process from the concept of operations. Because it is a work *process* it abstracts from work *practice*. The process consists of four main phases: Constraint Identification, Impact Assessment, Flow Planning and Flight Implementation. During these phases several parties collaborate to refine the routing and planning of flights.

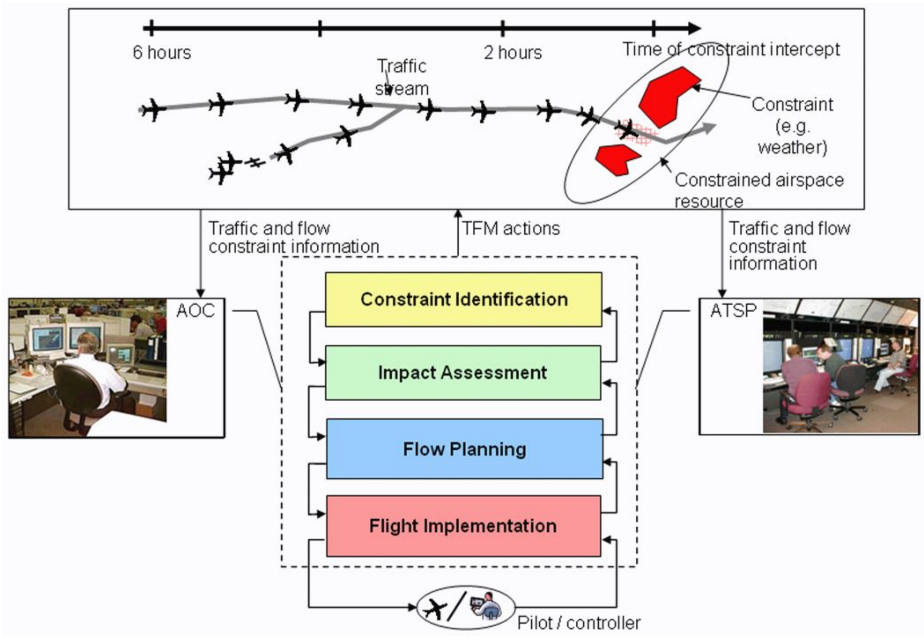

6 hours
2 hours
Time of constraint intercept
Traffic stream
Constraint (e.g. weather)
Constrained airspace resource
Traffic and flow constraint information
TFM actions
Traffic and flow constraint information
AOC
ATSP
Constraint Identification
Impact Assessment
Flow Planning
Flight Implementation
Pilot / controller

Fig. 1. The four main phases of the CTFM work process

At NASA Ames Research Center, this concept of operations is being evaluated through agent-based modeling and simulation, along with other CTFM concepts [22]. We want to know if the new, planned work processes are effective and feasible for the stakeholders, e.g., the FAA, airlines, and passengers. Most of the work processes have only been described on an abstract level (the organizational view), i.e., the desired outcomes of the processes. The specific implementation of the work processes that will lead to these outcomes still needs to be defined. This is not straightforward because it requires the coordination of many collaborative activities among highly specialized people in distributed and heterogeneous organizations. Additionally, it is hard to prescribe an exact work process. People may deviate from the objectives (e.g., file the flight plan of each flight) and violate norms, which are constraints on, or specializations of the objectives (e.g., file the flight plan before the flight takes off). Thus, the work process is not necessarily the work practice. Therefore, investigating varied work practice implementations (i.e., the expected future work practice) and their performance on the organizational objectives supports the evaluation of the CTFM concept of operations. This way, we need to model both the organizational view resulting in a work process model and the emergent view resulting in a work practice model (figure 2). The work process model in our case study has been derived from Airline Operations Centers' (AOC) documented field observations [13], and modeled in OperA. The work practice model in our case study has been made up, simply to show that the behavior of individual workers may conflict with the overall desired behavior of the AOC.

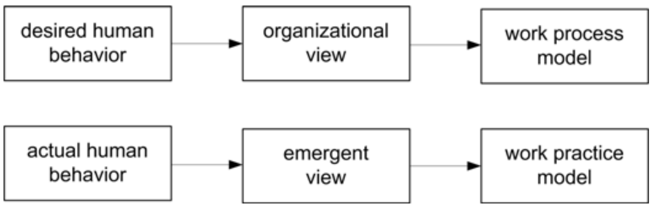


Fig. 2. The two views and the resulting models

3 OperA and Brahms

OperA models cannot be simulated without another framework for the specification of the agents' behavior. This is because OperA treats agents partly as 'black boxes', i.e., only the desired outcomes of their behavior are specified. However, the way in which these outcomes should be achieved is not specified. Because of that, an OperA model is not executable on its own. Brahms has been used to implement the agents' behavior. This is called an 'instantiation' or 'population' of the OperA model. This decoupling of the abstract description of the organization and the concrete description of the individuals is useful, because it is in accordance with reality, where different groups of people with different work styles may achieve the same objectives in different ways.

3.1 OperA

The OperA model for agent organizations enables the specification of organizational requirements and objectives, and at the same time allows workers to have the freedom to act according to their own capabilities and demands [5]. An OperA model can be seen as a recipe for collective activity; organizations are described in terms of roles, their dependencies and groups, interactions and global norms and communication requirements. Given that OperA assumes organizations as being open systems, it does not include constructs to the specification of the actual agents, treating them as 'black boxes' that commit to a specific (negotiable) interpretation of the organizational roles. OperA meets the following requirements:

- **Internal autonomy requirement:** The internal behavior of the participating agents should be represented independently from the structure of the society.
- **Collaboration autonomy requirement:** The external behavior of the participating agents should be specified without completely fixing the interaction possibilities in advance.

Table 1. Overview of OperA methodology

	Step	Description	Result
OM	Coordination Level	Identifies organization's main characteristics: purpose, relation forms	Stakeholders, facilitation roles, coordination requirements
	Environment Level	Analysis of expected external behavior of system	Operational roles, use cases, normative requirements
	Behavior Level	Design of internal behavior of system	Role structure, interaction structure, norms, roles, scripts
SM	Population Level	Design of enactment negotiation protocols	Agent admission scripts, role enactment contracts
IM	Interaction Level	Design of interaction negotiation protocols	Scene script protocols, interaction contracts

The OperA framework consists of three interrelated models. The **Organizational Model (OM)** is the result of the observation and analysis of the domain and describes the desired behavior of the organization, as determined by the organizational stakeholders in terms of objectives, norms, roles, interactions and ontologies. The **Social Model (SM)** maps organizational roles to specific agents. Agreements concerning the roles an agent will play and the conditions of the participation are described in social contracts. The **Interaction**

Model (IM) specifies the interaction agreements between role-enacting agents as interaction contracts.

A generic methodology to determine the type and structure of an application domain is described in [5]. Organizational design starts from the identification of business strategy, stakeholders, their relationships, goals and requirements. It results in a comprehensive organizational model including roles, interactions, objectives, and norms, which fulfill the requirements set by the business strategy. A brief summary of the methodology is given in 1.

There are many dimensions that can be included or excluded from organizational modeling. We have compared OperA with several other organizational modeling languages on the following five dimensions:

- *Structural*: what the organization consists of, e.g. groups, roles
- *Functional*: what the organization wants to achieve: objectives, procedures
- *Behavioral*: what the individuals in the organization actually do: activities
- *Communicative*: also called 'dialogical' or 'ontological'; how individuals in the organization communicate
- *Normative*: also called 'deontic'; how behavior within the organization can be controlled

Most organizational models (e.g., AGR [10]) used to focus on the structural and functional dimensions. Recently, models have also started to incorporate the normative dimension (e.g., Moise+ [12]). A few models also include the communicative dimension (e.g., OMNI [8]). OperA mainly covers the structural, functional and normative dimensions. In our opinion, the behavioral dimension, which is related to the emergent view, is underexposed in most organizational modeling languages [19]. In section 6 we will go into more detail on related work in this respect.

3.2 Brahms

Modeling and simulating work processes is often done at such an abstract level that individual work practice, such as collaboration, communication, 'off-task' behaviors, multi-tasking, interrupted and resumed activities, informal interactions, use of tools and movements, is left out, making the description of how the work in an organization actually gets done impossible. The Brahms modeling language is geared towards modeling people's activity behavior, making it an ideal environment for simulating organizational processes at a level that allows the analysis of the work practice and designing new work processes at the implementation level [5,17].

The Brahms framework consists of several interrelated models. The **Agent Model** describes the behavior of individuals (i.e., people) and groups of individuals (i.e., communities of practice). Individuals are members of groups and inherit the behavior of the groups. Individuals can also have additional behavior that distinguishes them from other individuals, and they can be a member of multiple groups (i.e., multiple inheritance). Groups can be organized in a hierarchical way, to define behavior at different levels of abstraction. Sub-groups inherit the behavior of super-groups. This is convenient for the modeling of common objectives and activities, and does not limit the agents' autonomy because anything specified on the group level can be overloaded on the agent level. The **Object Model** describes non-cognitive objects (i.e., things). Objects can be physical, or conceptual. The latter means that they only exist within the minds of agents, and can therefore not influence and react on the world. The **Knowledge Model** describes the reasoning of agents and objects, which is based on beliefs and facts. Beliefs are propositions that represent the world state and are internal to the agent or object. Facts are actual world states, and are global in the simulation world. The **Activity Model** defines the behavior of agents and objects by means of activities and workframes. Brahms has an activity-based subsumption architecture by which an agent's activities can be decomposed into sub-activities. Activities can be interrupted and resumed, just as humans can multitask by switching between different activities. Workframes control when activities are executed based on the beliefs of the agent, and on facts in the world. The **Communication Model** defines communication activities between agents and objects. When an agent or object communicates, it either sends or receives beliefs from other agents or objects. The **Geography Model** defines a hierarchy of geographical locations representing the space where activities occur. Agents and objects are located in areas and can move from area to area, possibly carrying other agents or objects, by performing a move activity.

There are several requirements for a work practice modeling language. First, it should be a simulation language, i.e., a language that supports the modeling of time. Second, it should support the modeling of activities rather than goals. Third, it should support subsumption, and reactive behavior. Brahms fulfills these requirements because it is a BDI-like activity language. Brahms differs from Jack and Jade in that Brahms is a compiled declarative agent-oriented language. Brahms differs from Jason in that Jason is a goal-based language, while Brahms is an activity-based language. Jason agents are represented using prescribed problems and plans to solve them. Brahms is a behavioral BDI language based on a reactive subsumption architecture, where competing activities are active at once on multiple levels. This allows for seamless activity switching, based on context information the agent is aware of (i.e., has beliefs about). For a description of different multi-agent languages see [1]. For a discussion of how the Brahms language differs from other BDI languages, see [16].

3.3 Rationale for Integration of OperA and Brahms

Although OperA was developed almost 10 years later than Brahms, the **philosophy** behind OperA and Brahms is similar. Brahms was developed because work processes were often modeled too abstract, i.e., formal descriptions of work processes differed too much from the actual work practice. Similarly, OperA tries to bridge the gap between the official and the real-world. However, OperA and Brahms have a different **viewpoint** on the solution to this problem. Brahms tries to bridge the gap between the abstracted and the real work practice bottom-up, i.e., by observing and describing the individual behavior of people. The modeler can then observe what collective behavior emerges from the interaction of the individual behavior of the people: the emergent view on agent-based modeling. OperA tries to bridge the gap top-down, by describing the objectives of an organization. This way it defines what the result of the emergent behavior of the collective should be, rather than describing the practice (i.e., the individual activities and interactions) that should lead to that result: the organizational view on agent-based modeling.

The difference in the viewpoints becomes clear when we compare the **models** that result from the different methodologies. Brahms mainly consists of agents that reason (Knowledge Model) and work (Activity Model). These are definitions of the work that gets done, rather than the results that should be achieved. OperA defines roles, objectives, and norms (Organizational Model). It also defines social contracts (Social Model), which allow the modeler to define which particular agent executes which roles, and which special norms apply. Finally, it defines interaction contracts (Interaction Model), which allow the modeler to describe norms that apply when two or more specific agents, enacting specific roles, interact. These are definitions of (the restrictions on) the results that should be achieved, rather than definitions of the process itself. This shows that OperA and Brahms are orthogonal in this respect.

OperA and Brahms are different **languages**. Brahms is an implementation language. It is formal, and can be compiled to Java, and executed using the Java virtual machine. OperA is a conceptual language, which level of formality depends on the preferences of the modeler and on the development state of the model. Modeling can start by defining objectives and norms in terms of natural language, and then moving gradually to pseudo-logic and finally to deontic logic. OperA semantics are formally grounded on the temporal deontic logic LCR [6].

While Brahms is mainly a language, OperA is more of a **methodology** because it provides guidelines on how to get from abstract definitions of work processes (i.e., objectives and norms) to more specific definitions of work processes (i.e., social contracts). This way, there is an order in the models that are created, while in Brahms this is completely up to the modeler.

4 Integration of OperA and Brahms

Based on the complementary viewpoints of OperA and Brahms described in the previous section, we hypothesized that, after integration, the two frameworks could complement each other in the following two ways: (1) OperA adds the organizational (top-down) view to Brahms, Brahms adds the emergent (bottom-up) view to OperA, so that both perspectives are represented, (2) simulations can be run that show the difference between the two perspectives (i.e., the normative gap). Figure 3 shows our WSMS methodology, which consists of several steps that are needed to realize point 2. The next section explains this figure.

4.1 Methodology

Figure 3 is a schematic view of our WSMS methodology. The squares represent models, the squares with rounded corners represent actions of the modeler (i.e., methods). The operational concept can be defined in any formal or informal way (see the OperA methodology [5]. It then needs to be defined in OperA during the organizational modeling action, resulting in the organizational model in OperA. It is also possible that the operational concept only exists in the mind of the modeler and is modeled in OperA right away. At the same time, before, or after organizational modeling, the work practice should be modeled, resulting in the work practice model in Brahms. The work practice can be defined in any formal or informal way and can then be defined in Brahms, or it can only exist in the mind of the modeler (e.g., after observation of human behavior) and be defined in Brahms right away.

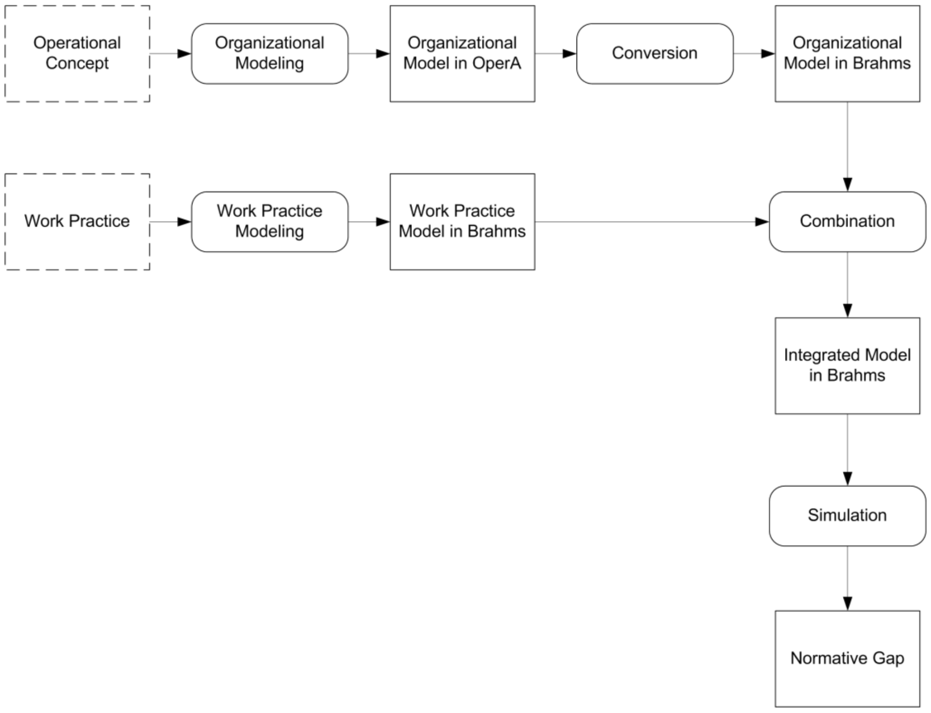


Fig. 3. Work Systems Modeling and Simulation methodology

Because both models need to be described in Brahms in order to be executable together, the organizational model in OperA needs to be converted to Brahms. Currently, this is done manually, using the mappings, but it seems realistic that this can be automated. Currently, OperettA, a tool for automated verification of OperA models is being developed and is very promising in this respect [7]. After conversion, both models are defined in Brahms.

Although both levels are now defined in the same language, they may not yet be totally integrated due to possible structural and ontological differences. Therefore a combination method is required, which requires some interpretation of the modeler. This method is out of the scope of this paper, but is described in more detail in [21].

Finally, the two layers are integrated and the simulation can be run. The output of the simulation will show which organizational objectives have been achieved, or not, and which norms have been violated. Additionally, it will show which agents were involved and the context of the compliance or violation situation.

4.2 Conversion

OperA consists of three main models: OM, SM and IM. Each of these models is further subdivided into levels and structures (cf. table 1). If we break these constructs further down we get OperA's atomic constructs, some of which have been listed in Table 2. In order to be able to convert an OperA model to a Brahms model, we have defined Brahms equivalents for each of the OperA constructs (also in Table 2). Sometimes an OperA construct can be represented by a single, simple Brahms construct, other times several interrelated constructs are needed. Currently, we have defined almost all mappings, without any major difficulties.

Space and scope limitations do not allow for an in-depth discussion of all mappings, for which we refer to [19]. We suffice with an example of one of them, to show that the mapping meets the internal autonomy requirement. Row 'Objective' in table 2 shows a part of the OperA role definition of the dispatcher, which is a common role in Airline Operation Centers [13]. Dispatchers' main task is to ensure the safety of flights. The code shows that the objective is defined on the role level by means of a protocol, which is a workframe that is executed by all agents that are members of the Dispatchers group. However, agents can decide to ignore the protocol and implement a different kind of behavior. Agent Diana is an example of an agent that decides to use her autonomy: she supervises Dave and she knows that she only needs to ensure the safety when Dave is not available. This is the kind of information that is often abstracted from in work process models, but which is included in work practice models. This mapping meets OperA's internal autonomy requirement, as it allows the designer to define the individual behavior of agents independently from the desired behavior that is defined on the role (organizational) level. The protocol is a useful mechanism to prevent code duplication in the case of common behavior among multiple agents. By making those agents members of one group with a protocol, the protocol's workframe only needs to be defined once.

Table 2. Overview of OperA constructs with Brahms equivalents

OperA	**Brahms**

Role Enactment
Dispatcher $\in$ Roles$_O$
	group 'roles' with sub-groups for each converted OperA role
	`group Roles {…}`
	`group Dispatchers memberof Roles {…}`

Group Membership
AOC $\subseteq$ Groups$_O$
	group 'groups' with sub-groups for each converted OperA group
	`group Groups {…}`
	`group AOC memberof Groups {…}`
	`group Dispatchers memberof Roles, AOC {…}`

Objective
safety_ensured (Flight) $\in$ Objectives$_{Dispatcher}$
	protocol in converted OperA role + e.g., overloading in agent
	`group Dispatchers memberof Roles, AOC {`
	`  composite_activity`
	`ca_safety_ensured(Flight flight) {`
	`    // protocol to ensure safety  }`
	`  workframe wf_safety_ensured {`
	`    when(knownval(current.followProtocols =`
	`true)) do { ca_safety_ensured(flight); }}}`
	
	`agent Diana memberof Dispatchers {`
	`  initial_beliefs:`
	`  (current.followProtocols = false);`
	`  …`
	`  workframe wf_safety_ensured {`
	`    when (knownval(Dave.available = false))`
	`    do { ca_safety_ensured(flight); }}}`

Role Norm
IF taken_off (Flight) THEN OBLIGED released(Flight)
	workframe in agent Monitor
	`agent Monitor {`
	`  workframe wf_nv_flight_released {`
	`    when(knownval(flight.taken_off = true)`
	`and knownval(dispatcher responsibleFor`
	`flight) and not(flight.released = true))`
	`    do { co_norm_violation(violation); }}}`
	
	`// complying agent`
	`agent Diana memberof Dispatchers {`
	`  workframe wf_flight_released {`
	`    when(knownval(current responsibleFor`
	`flight)) do { ca_flight_released(Flight`
	`flight); }}}`
	
	`// violating agent`
	`agent Dave memberof Dispatchers {`
	`  workframe wf_email_read {`
	`    do { pa_email_read(); }}}`

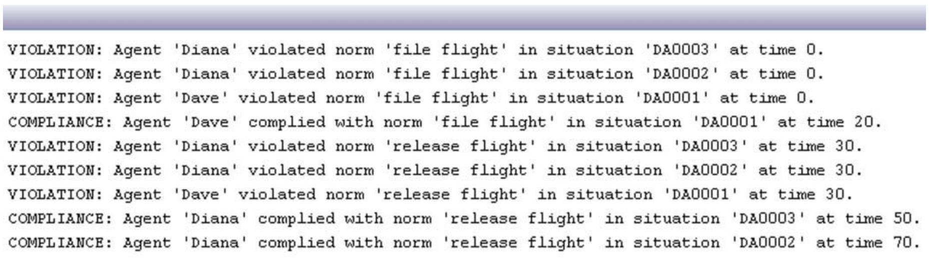
Group Norm
OBLIGED attended (Meeting)
	workframe in agent Monitor
	`agent Monitor {`
	`  workframe wf_nv_meeting_attended {`
	`    foreach(AOC) AOC_member;`
	`    foreach(Meeting) meeting;`
	`    when(not(meeting attendedBy`
	`AOC_member))`
	`    do { co_norm_violation(violation); }}}`

Social Contract Norm
SContract(Pete, Planner, {FORBIDDEN (contacted(Pilot))})
	workframe in agent Monitor
	`agent Monitor {`
	`  workframe wf_nv_pilot_contacted {`
	`    foreach(Pilot) pilot;`
	`    when(knownval(pilot contactedBy Pete))`
	`    do { co_norm_violation(violation); }}}`

Interaction Contract Clause
IContract(Diana, Pete, Scene, {…})
	workframe in agent Monitor
	`agent Monitor {`
	`  workframe wf_nv_plan_revised {`
	`    foreach(Plan) plan;`
	`    when(knownval(plan authorizedBy Pete)`
	`      and not(plan revisedBy Diana))`
	`    do { co_norm_violation(violation); }}}`

5 Case Study: Validation

The integration of OperA and Brahms makes it possible to run a simulation in Brahms in which both the organizational view (e.g., roles, objectives, norms) and the emergent view (e.g., activities, workframes) are represented. But these two views are not necessarily aligned: the work practice may differ from the intentions of the organization's policy makers. Objectives may not be met, and norms may be violated. We have therefore extended the simulation with a monitoring agent ('agent Monitor'), which can detect norm violations. (This is the simplest way, but in the future we would like to support different types of organizations that require agents to monitor themselves or each other.) This makes it possible to perform norm-based evaluation of the proposed CTFM operational concepts, which are usually described from the organizational perspective.



```
VIOLATION: Agent 'Diana' violated norm 'file flight' in situation 'DA0003' at time 0.
VIOLATION: Agent 'Diana' violated norm 'file flight' in situation 'DA0002' at time 0.
VIOLATION: Agent 'Dave' violated norm 'file flight' in situation 'DA0001' at time 0.
COMPLIANCE: Agent 'Dave' complied with norm 'file flight' in situation 'DA0001' at time 20.
VIOLATION: Agent 'Diana' violated norm 'release flight' in situation 'DA0003' at time 30.
VIOLATION: Agent 'Diana' violated norm 'release flight' in situation 'DA0002' at time 30.
VIOLATION: Agent 'Dave' violated norm 'release flight' in situation 'DA0001' at time 30.
COMPLIANCE: Agent 'Diana' complied with norm 'release flight' in situation 'DA0003' at time 50.
COMPLIANCE: Agent 'Diana' complied with norm 'release flight' in situation 'DA0002' at time 70.
```

Fig. 4. Output of a simulation (the virtual machine's log)

The code in row 'Role Norm' in table 2 shows the conversion of an OperA norm into Brahms code. The Monitor checks if there are dispatchers who are responsible for flights, that have taken off, but that have not yet been released. If this is the case, a norm violation occurs, because normally flights are released before they take off. Agent Diana complies with the norm by releasing the flight. Agent Dave is reading e-mail and is therefore not able to comply with the norm. When a norm is violated, an object is created that contains the identifier of the norm, the situation in which it has been violated (e.g., for which flight), the moment in time, and the agent by which it has been violated. The monitoring agent creates the objects, and reports them to the user of the simulation (e.g., via the virtual machine's log: figure 4). The user can then determine the frequency and type of the violations, and which agents are more likely to violate which norms. These simulations show to what degree the actual work practice differs from the organizational objectives (i.e., the work process model). Eventually, this information can be used by policy makers and workers themselves to change organizational objectives and norms, or to improve the work system.

6 Related Work

In this research we integrate work process modeling with work practice simulation. To our knowledge this is a novel approach for the specification of complex distributed environments where cooperation is necessary and the entities are autonomous. For this effect, we used two modeling languages: OperA and Brahms. Both are existing, exemplary, frameworks for their own class of modeling paradigms: OperA for organizational modeling and Brahms for work practice simulation.

In section 3.2 we have discussed how Brahms differs from other agent languages, rather than work practice languages, because to the best of our knowledge, there is no other work practice modeling and simulation language. Replacing Brahms with another agent language (and defining new mappings from OperA to this language), could still lead to an integration of the organizational and emergent views. However, for work practice modeling and simulation, the agent language would have to be BDI-based, declarative, activity-based, and support subsumption [19].

In section 3.1 we have shortly discussed how OperA differs from other organizational modeling languages. OperA is an organizational model that *supports* (rather than *incorporates*) the emergent view because it allows the specification of individuals in another implementation language and does not put any requirements on the internal specification of those agents. An organizational model that *incorporates* the emergent view allows the specification of the internal behavior of individuals, independently of the specification of organizational policies.

Some other languages, such as HarmonIA [20], OMNI [8] and RNS2 [21], also *support* the emergent view and may replace OperA in that respect. Approaches like ISLANDER [9], do not support nor incorporate the emergent view because either the agent population is specifically generated to fulfill the norms of the organization or because there is a control mechanism that blocks all disallowed activities. Although one could argue that there is low level behavior that emerges, the behavior is fully predictable, which is useful for the modeling of electronic institutions, but is not suitable for work practice modeling.

S-Moise+ [11] even *incorporates* the emergent view by means of the J-Moise+ agent architecture. However, this is a goal-based architecture rather than an activity-based architecture, while an activity-based architecture is required for work systems modeling [16]. RNS2 is activity-based, however can only be used to specify activities on the role level, which means that unique human individuals cannot be described. Therefore this language cannot be used to describe *actual* behavior and is thus not suitable for work systems modeling.

7 Conclusion

The design and evaluation of work systems can be supported with agent-based modeling and simulation that incorporates both an organizational view (top-down) and an emergent view (bottom-up) on work systems. Most current modeling and simulation frameworks only focus on either one of these views. Therefore we have integrated two frameworks, each representing one of the two views. The combined Work Systems Modeling and Simulation (WSMS) framework makes it possible to simulate work practice, and to monitor the gap between the emergent behavior and the desired outcomes as defined by the organization's policy makers. Our initial WSMS methodology has been applied to the CTFM concept of operations, by investigating which population and specification of agents' activities and interactions leads to the desired objectives, and conversely, which objectives can be met, based on a certain work practice.

The hypotheses for this research were that the integration of OperA and Brahms meets the following requirements: (1) OperA adds the organizational (top-down) view to Brahms, Brahms adds the emergent (bottom-up) view to OperA, so that both perspectives are represented, (2) simulations can be run that show the difference between the two perspectives (i.e., the normative gap). The work presented in this paper provides a proof of concept that supports the validity of these hypotheses. However, more work is needed to improve the integrated methodology by applying it to different cases, and to determine the actual usefulness of norm-based evaluation of operational concepts.

This work contributes to our more general research objective: How can we improve models of work practice by incorporating the organizational view, What happens when agents become aware of the fact that they are violating a norm?, What is the influence of norms on work practice?, and: How do norms arise from work practice? New insights in these areas will lead to more realistic models of work practice, and thereby to improved agent-oriented system engineering methodologies.

Acknowledgements

This material is based on work supported by NASA under award NNA07BB97C. The research of V. Dignum is funded by the Netherlands Organization for Scientific Research (NWO), through Veni-grant 639.021.509.

References

1. Bordini, R., Dastani, M., Dix, J., Seghrouchni, A.: Multi-Agent Programming: Languages, Platforms and Applications. Springer, Heidelberg (2005)
2. Clancey, W.J., Sachs, P., Sierhuis, M., Hoof, R.V.: Brahms: Simulating practice for work systems design. International Journal on Human-Computer Studies 49, 831–865 (1998)
3. Colvin, A., Boswell, W.: The problem of action and interest alignment: Beyond job requirements and incentive compensation. Human Resource Management Review 17, 38–51 (2007)
4. Dignum, V., Dignum, F., Jonker, C.: Towards agents for policy making. In: David, N., Sichman, J. (eds.) MABS@AAMAS 2008 (2008)
5. Dignum, V., Dignum, F., Meyer, J.: An agent-mediated approach to the support of knowledge sharing in organizations. Knowledge Engineering Review 19(2), 147–174 (2004)
6. Dignum, V., Meyer, J., Dignum, F., Weigand, H.: Formal specification of interaction in agent societies. In: Hinchey, M.G., Rash, J.L., Truszkowski, W.F., Rouff, C.A., Gordon-Spears, D.F. (eds.) FAABS 2002. LNCS (LNAI), vol. 2699, pp. 37–52. Springer, Heidelberg (2003)

7. Dignum, V., Okouya, D.: Operetta: A prototype tool for the design, analysis and development of multi-agent organizations. In: Proc. AAMAS 2008, Demo Track (2008)
8. Dignum, V., Vzquez-salceda, J., Dignum, F.: Omni: Introducing social structure, norms and ontologies into agent organizations. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2004. LNCS, vol. 3346, pp. 181–198. Springer, Heidelberg (2005)
9. Esteva, M., Cruz, D., Sierra, C.: Islander: an electronic institution editor. In: Proc. AAMAS 2002 (2002)
10. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: An organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
11. Hübner, J., Sichman, J., Boissier, O.: S-moise+: A middleware for developing organised multi-agent systems. In: Boissier, O., et al. (eds.)COIN I. LNCS (LNAI), vol. 3913, pp. 64–78. Springer, Heidelberg (2006)
12. Hübner, J., Sichman, J., Boissier, O.: Developing organised multi-agent systems using the moise+ model: Programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering 1(3/4), 370–395 (2007)
13. Idris, H., Evans, A., Vivona, R., Krozel, J., Bilimoria, K.: Field observations of interactions between traffic flow management and airline operations. In: 6th AIAA Aviation, Technology, Integration, and Operations Conference (2006)
14. Idris, H., Vivona, R., Penny, S., Krozel, J., Bilimoria, K.: Operational concept for collaborative traffic flow management based on field observations. In: 5th AIAA Aviation, Technology, Integration, and Operations Conference (2005)
15. Sierhuis, M.: Modeling and Simulating Work Practice; Brahms: A Multiagent Modeling and Simulation Language for Work System Analysis and Design. SIKS Dissertation Series 2001-10. University of Amsterdam, Ph.D Thesis (2001)
16. Sierhuis, M.: It's not just goals all the way down - it's activities all the way down. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS (LNAI), vol. 4457, pp. 1–24. Springer, Heidelberg (2007)
17. Sierhuis, M., Clancey, W., van Hoof, R.: Brahms: A multiagent modeling environment for simulating work processes and practices. International Journal of Simulation and Process Modelling 3(3), 134–152 (2007)
18. van der Vecht, B., Dignum, F., Meyer, J.-J., Neef, M.: A dynamic coordination mechanism using adjustable autonomy. In: Sichman, J.S., Padget, J., Ossowski, S., Noriega, P. (eds.) COIN 2007. LNCS, vol. 4870, pp. 83–96. Springer, Heidelberg (2008)
19. van Putten, B.-J.: Opera and brahms: a symphony? integrating organizational and emergent views on agent-based modeling and simulation. Technical Report INF/SCR-07-79, Utrecht University (2008)
20. Vazquez-Salceda, J.: The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains. The HARMONIA framework. Ph.D thesis, Universitat Politecnica de Catalunya (2003)
21. Weiss, G., Nickles, M., Rovatsos, M., Fischer, F.: Specifying the intertwining of cooperation and autonomy in agent-based systems. International Journal of Network and Computer Applications 30(3), 1196–1215 (2007)
22. Wolfe, S.: Supporting air traffic flow management with agents. In: AAAI Spring Symposium: Interaction Challenges for Intelligent Assistants. AAAI, Menlo Park (2007)

Support for Analysis, Design, and Implementation Stages with MASDK

Vladimir Gorodetsky, Oleg Karsaev, Vladimir Samoylov, and Victor Konushy

St. Petersburg Institute for Informatics and Automation
39, 14 Liniya, St. Petersburg, 199178, Russia
{gor,ok,samovl,kvg}@mail.iias.spb.su

Abstract. In spite of much research and development on agent-oriented software engineering methodologies and supporting software tools, the problem remains of topmost importance. Many efforts are still needed to make such methodologies and software tools practically applicable at an industrial scale. This paper proposes extension of the Gaia methodology with a formal specification language, making it possible to implement Gaia as a model-driven engineering process supported by a corresponding agent-based software development environment, MASDK 4.0. The paper outlines MASDK 4.0 through the extended Gaia, and demonstrates the technology supported by MASDK 4.0 on the basis of a fragment of a case study on autonomous air traffic control.

1 Introduction

Over the last decade, different tools, software libraries, frameworks and program languages [1] have been developed for multi-agent system (MAS) development. Among them, the most widely used are Living SystemsRTechnology Suite [18] AgentBuilder [17], agentTool [5], Coguaar [8], JADE [2], INGENIAS IDE [12], etc. These differ in methodologies used (MaSE [6], Tropos [11]), Gaia [20], etc.), architecture of target applications (BDI, reactive architecture, etc.), and in levels of maturity achieved and classes of applications they are able to develop. However, there is no single methodology and software tool that is the best one.

Among other agent-oriented software engineering methodologies, the main peculiarity of Gaia is that it is not explicitly goal-oriented, although this feature does not mean that goal-oriented MAS applications of BDI architectures cannot be developed on the basis of Gaia. It focuses on organizational abstractions of applications with subsequent explicit separation and specification of internal and external behaviour of agents composing MAS, and exactly this feature determines its specificity as opposed to many other existing methodologies. External behaviour determines agent interactions in various use cases that is specified in terms of interaction protocols and constrained by organizational rules. Explicitly introduced protocols actually make it much simpler to represent collective behaviour of agents, and the behaviour itself is more predictive in comparison with the goal-oriented BDI approach. In the BDI approach, the emerging behaviour of individual agents and collective behaviour results from rich knowledge and reliable beliefs of the particular agents, as well as from sophisticated inference mechanisms that need to use, as a formal basis, modal and temporal logic. In many cases, the BDI approach works well, but the problem of complexity should be managed carefully.

In contrast, Gaia attempts to transfer MAS development complexity into more careful analysis of the system's organizational issues, and the explicit design of the MAS interaction model thus makes collective behaviour of agents more predictable and understandable. In general, such an approach simplifies agent-oriented software engineering and there exist many classes of applications where Gaia-based technology could outperform purely goal-oriented ones.

The objectives of this paper are twofold. One is to extend Gaia in order to make it more applicable by enriching it with a formal specification language that can make it possible to practically implement Gaia as model-driven engineering. The second objective is, based on the extended Gaia and a design process and formal specification language, to introduce the MAS development environment that implements a graphical MAS development process, according to the extended Gaia methodology. Accordingly, the paper is organized as follows. Section 2 presents a general view of the Multi-agent System Development Kit (MASDK 4.0) components and their interactions in the development procedure. Section 3 sketches Gaia as it was proposed by the authors [20] and presents generally in what aspects it is extended in this paper. Section 4 introduces the Agent System Modeling language, ASML, that supports model-driven engineering technology of the Gaia implementation. Section 5 describes the stage-by-stage development process supported by MASDK 4.0 bridging it with Gaia and describing the particular products. This is done using a fragment of a case study for an autonomous air traffic control system. Section 6 surveys related work on the existing extensions of Gaia. The conclusion summarizes the main paper contributions, emphasizes the advantages of the proposed Gaia extensions and the MASDK 4.0 development environment implementing Gaia. Future work is also outlined.

2 Architecture of MASDK 4.0 Environment

MASDK 4.0 is a software tool implementing an extended version of the Gaia methodology. It is provided with a graphical and formal specification language that supports thorough and consistent conceptual analysis, detailed design, code generation and deployment of target multi-agent applications, including those operating in dynamic and Peer-to-Peer (P2P) environments.

The basic components of MASDK 4.0 and their interaction are shown in Fig. 1. The *Agent-based System Modeling Language* (ASML), based on UML, is exploited for in-progress-design specification of MAS formal models. ASML-based specification of MAS is supported by a *Visual Design Environment* (VDE) component providing for user-friendly graphical notation of ASML. The final formal model or model-in-progress is stored in the *MAS Model Description* (MMD) component. The resulting MAS formal model serves as input for producing the agent classes' source code. This functionality is performed by the *Source Code Builder* (SCB) component. Two more components, *Generic Agent* (GA) and *Agent Platform* (AP), are implemented as reusable solutions (source code). The code of the GA component realizes application-independent functionality of agents behaviour, implemented as a reusable library producing the agent classes' source code. The AP is FIPA compliant developed according to the reference model [8], and realizes distributed "white" and "yellow" pages services and provides

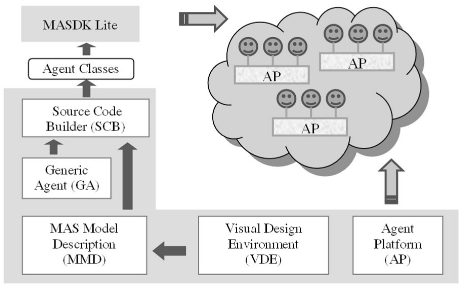

MASDK Lite
Agent Classes
Source Code Builder (SCB)
Generic Agent (GA)
MAS Model Description (MMD)
Visual Design Environment (VDE)
Agent Platform (AP)
AP
AP
AP

Fig. 1. MASDK environment components and their interaction

for communication. The AP is a self-dependent software component that can be exploited separately from MASDK 4.0. Its description and downloadable run-time code can be found in [15].

MASDK Lite is an auxiliary component intended for agent deployment. Specification of agent classes (source code of agent classes) is used as its input. The agent instances deployment requires identification of their names and addresses, and, when necessary, specification of their initial mental models and services they possess. MASDK Lite, like AP, can be used separately from MASDK 4.0 for application maintenance.

MASDK 4.0 supports analysis, architectural and detailed design of MAS applications by Gaia, but also software implementation using the SCB component, and deployment using MASDK Lite. The ASML-based formal specification of MAS models is used by the SCB component to produce source code of agent classes. The source code contains fully specified behaviour logic and mental models of agent classes. Therefore, the implementation stage is limited to encoding of the agent class activities. This code should be manually developed by programmers using the MS VC++ environment.

3 Methodological Basis: Extended Gaia

As stated above, the Gaia methodology for agent-oriented software engineering [20] is focused on organizational abstractions of complex systems with explicit separation and specification of internal and external behaviours of agents. External behaviour determines agent interactions in various use cases, specified by interaction protocols. Interaction protocols are distributed algorithms performed by a subset of agents each of which performs, within particular protocol, a specific role. The scenario of the role performance and some other role activities correspond to what is above called "internal agent behaviour". Accordingly, Gaia explicitly determines the decomposed system organization components, the subtasks to be solved by each component in various use cases (roles), and interaction protocols associated with the use cases. The results of the Gaia analysis constitute the input of the subsequent architectural and detailed design.

In fact, Gaia, as described by the authors [20], specifies a general methodological framework for analysis and architectural design, whereas methodology applicability needs detailed development and formal support for all aforementioned stages and implementation, and deployment issues as well. In other words, applicability of Gaia requires its technology-oriented extension. Extensions of various kinds were proposed in [10], [3], [13], [14], etc. reviewed in Section 6. This section outlines the applicability-oriented Gaia extensions proposed in this paper that then constitute the methodological basis for the agent-oriented software engineering technology fully supported by MASDK 4.0 components presented in Fig. 1.

3.1 Analysis

Subdivide System into Sub-organizations. The objective of this stage is description and analysis of the organization aiming at discovery of appropriate decomposition of the whole organization into more or less weakly-coupled sub-organizations.

Identify Environmental Entities. This activity aims at detection of environment conditions, constraints and environment entity interfaces. Examples of entities include databases, external services, user interface, sensors (e.g. controllers of an assembly line), etc.

Discover Roles, Create their Preliminary Internal and Interaction Models. Role discovery is fulfilled by an expert. Preliminary role model creation includes description of their *permissions* (e.g. regarding interaction with environment, etc.), *responsibilities and activities* together determining the role internal behaviour. The responsibilities are specified by *Liveness Expressions* (LE) using the formal language *Fusion*. The discovered roles are represented by the *Role Schemata*. Role interaction protocols are also identified. These models are preliminary and may be refined later.

Determine Organizational Rules. Organizational rules determine global system behaviour policy that must be satisfied by all the agents. E.g., security policy rules regulate access to confidential information. Safety policy rules in air traffic control determine admissible movements of the aircrafts in various situations when separation distances between aircrafts may be violated. In Gaia, organizational rules are introduced informally, constituting inputs for the architectural design stage.

An extension of the Gaia analysis proposed regards using a specific formal specification language, *Agent-based System Modeling Language* (ASML), supporting all the analysis-related activities and specifying formally the results. It is described below. These extensions are fully supported by MASDK 4.0.

3.2 Architectural Design

Gaia determines two kinds of activities that should be performed at this stage.
Select Organizational Structure. Decomposition is a subject of the analysis stage, while organizational structure is developed at the architectural design stage so that various sub-organizations can be structured differently. E.g., some components can be structured in P2P mode, whereas others can be hierarchical.
Final Role and Interaction Models. The organizational structures can require refining of the preliminary role schemata (i.e., models of roles and interaction protocols).

The proposed Gaia extension supports formal specification of the design results using ASML and supported by MADK 4.0 (see below).

3.3 Detailed Design

The core of this stage is identification and detailed design of agent class models and services. Each agent class is allocated one or several identified roles. The products should correspond to a fully developed agent architecture, including its services, providing the specifications necessary for software implementation.

Agent Model. All the agents allocated the same roles determine a particular agent class. Detailed design of agent class software architectures is the main subject of this stage in Gaia, but Gaia gives no concrete methodology for this activity.

This paper extends Gaia in two aspects. First, it proposes the use of ASML for specification of the in-progress and final products of the agent model detailed design and, second, it proposes a *generic agent* architecture implemented as reusable software within MASDK 4.0.

Service Model. Gaia defines agent services vaguely. In this paper, an agent service is understood as a single activity or a sequence of activities (a scenario) that provides an agent with functionalities that do not necessarily fall into the block of functions invoked by an agent interaction protocol. They can be composed of LEs, proactive behaviour invoked by internal agent state and/or by the state of the environment (e.g. when agent requests for resources from environment).

Implementation and deployment. These very important stages are outside the scope of Gaia. The proposed extension, and the corresponding software engineering means implemented in MASDK 4.0 are described below.

4 Introduction to ASML Language

The ASML language is based on UML notation. It is supported by a user interface providing, for ASML, a graphic notation that allows representing and editing of MAS models via the VDE component (Fig. 1) by creating diagrams formally representing in-progress and final analysis and design solutions. The set of diagram types, their interrelations and use at the respective stages of MAS design correspond to Fig. 2. Below we provide a brief outline of ASML.

MAS macro model diagram. This diagram type is specified using ASML concepts: *tasks, agent roles*, *protocols, active entities (components),* and *agent class.*

The notion of *task* is used for assigning names to use cases. *Agent role*, in the macro model, is identified by the name and template containing the list of LE names defining the role internal behaviour and also the list of identifiers of the rules triggering proactive agent behaviour[1] (triggering rules, for short). Each of these rules is linked to the LE it triggers. *Active entity* specifies the interface of agents to external components that are not agents. Examples of active entities are interfaces to sensors, effectors, etc. The macro model is constituted of instances of the above notions and the relations between them. The following types of relations are used in ASML: *task – is initiated by – triggering rule / functionality of active entity; LE – is initiated by – protocol / triggering*

[1] Here proactive behaviour is initiated by something other than a protocol.

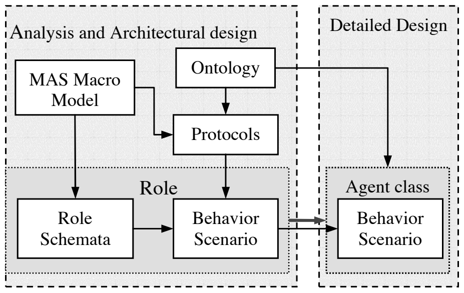


Fig. 2. Development process models and their interactions

rule; protocol – is <u>*initiated by*</u> *– active entity / Role / LE / functionality of active entity; functionality of active entity – is* <u>*initiated by*</u> *– protocol; agent class –* <u>*plays*</u> *– role.* Corresponding diagrams are intended for specification of the organizational structures and interactions, and a MAS application is specified by a single macro model that can contain several diagrams.

Protocol diagram. ASML provides the following protocol specification concepts: *(role) lifeline of the protocol participant, communication act, (role) lifeline end, alternative combined by a fragment* comprising two or more *operands*. Use of auxiliary dialogs allows specifying communication acts in terms of *cardinality of the protocol respondents, cardinality of the respondents* for each operand of combined fragments as well as for each message respondent. Communication act is specified in ACL language [9]. Specification of message content is done in terms of ontology concepts, while protocols are specified using the diagram editor. These diagrams specify interaction protocols among the roles and among the roles and active entities.

Role scheme. Detailed specification of each LE is performed at two abstraction levels using *Role scheme* and *behaviour scenario* diagrams respectively. The former is used to specify LE decomposition if necessary and scheme of inter-role behaviour coordination. *Role scheme* diagrams are represented in terms of such notions as *scenario, event, triggering rule* and *protocol.*

The LE decomposition is performed using two types of the designing rules, mandatory and optional. The former correspond to a case when the LE is linked to protocols or/and to triggering rules. In this case, the LE is decomposed into simpler scenarios with one-to-one relation to protocols, and triggering rules should be linked to a simple scenario. Optional rules allow experts to elaborate more detailed decompositions via adding new simple scenarios. The role scheme comprises scenarios reflecting LE decomposition and relations of type *scenario-*<u>*uses*</u>*-scenario.*

Each role's LE is run within a separate control thread that admits concurrency of role performance but this may require coordination. It is done using the notion of *event* and additional relations, i.e. *event –* <u>*is generated by*</u> *– scenario, event –* <u>*is used by*</u> *– scenario* and *event –* <u>*is used by*</u> *– triggering rule.* The relation of the first type identifies the behaviour scenario generating the corresponding event. The next type is intended for specification of the scenario interrupt processing. The same relation determines the events initiating continuation of the interrupt processing. The last type of the relation determines the conditions initiating triggering rules.

Role behaviour scenario. Diagrams of this type are used for specification of simple scenarios as a set of nodes and transitions between them, while representing behaviour logics of each scenario. ASML introduces the following self-explanatory types of notions defining classes of scenario nodes: *activity, complex activity, generating / waiting event, event handling, sending / waiting message, message handling, interface with agent platform, control node,* etc. E.g., in the nodes of type *control node*, different variants of the behaviour scenario performance continuations are specified. In the nodes representing a complex activity, the conditions invoking the behaviour embedded scenarios are determined, etc.

Ontology. Ontology diagrams are used for description of domain notions and relationships between them. Ontologies are used as the language for message content specification and, in the detailed design of agent classes, it is used to specify their behaviour scenarios in terms of model variables and attributes. Ontology relations are specified using three standard relations types: *generalization (inheritance), association and composition.*

Agent class behaviour scenario. When an agent class is allocated roles (at architectural design) the former inherits the roles' behaviour scenario list of these roles. Detailed design assumes formal specification of agent variables, scenario interface, scenario variables and scenario node calls represented using scenario and agent variables.

5 Support of Gaia in MASDK 4.0

5.1 Case Study: Autonomous Air Traffic Control

Due to the ever increasing intensity of air traffic and stiffening safety requirements, air traffic control relying on purely human-based control needs to reconsider its basic organizational principles. According to current opinion, some complex real time control responsibilities of air traffic control operators should be assigned to aircraft software to make control more autonomous. The case study of an agent-based autonomous air traffic control system, used for explanation of the developed software tool, MASDK 4.0, is outlined below.

The air traffic safety is provided by two measures. The first is structuring the airport airspace according to a topology that determines all the admissible trajectories of aircraft arrivals, landing and departure. It encompasses two zones (Fig. 3): (1) *arrival zone* and (2) *approach one.* The arrival zone comprises *arrival schemes*, which begin with entry points and are specified as sequences of legs ending with a *holding areas.* The airport airspace (AA) topology also determines admissible echelons, i.e. admissible altitude ranges for passing through leg exit points. The AA topology also contains departure schemes but departure control is omitted in the case study. An example of an AA topology for JFK airport of New York City (NYC) is depicted in Fig. 3.

The second measure of the air traffic safety provision assumes the separation standards that determine the minimal admissible distances between aircrafts along each of

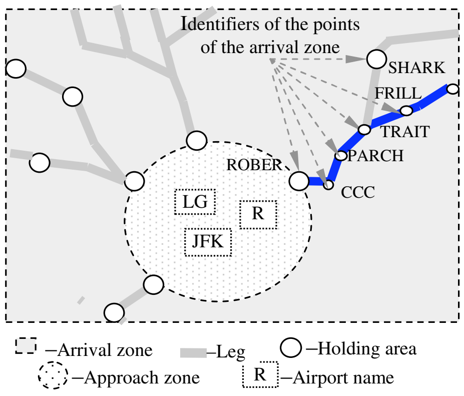


Fig. 3. Airspace topology within NYC area (horizontal projection), and arrival / approach zones

three spatial dimensions. These standards may be different for various air traffic-related situations. In the case study, meeting the separation standards is achieved by using a rule-based distributed safety policy that has to be followed by every aircrafts operating within airport airspace.

The idea of the autonomous air traffic control in the arrival zone is to delegate, to the aircraft's pilot-assisting software, the right to autonomously compute the safe landing trajectory, predict potential conflicts (violation of the separation standards) with other aircraft and to autonomously resolve these conflicts using a distributed safety policy and peer-to-peer negotiations with the potentially conflicting aircraft.

5.2 MASDK 4.0 Products Supporting Gaia

The whole air traffic control system (ATC) organization contains two sub-organizations: ATC in arrival zone and ATC in approach zone. Below, the latter sub-organization is not considered. The environmental model includes the model of the AA topology, the real time model, and visualization model of the whole situation in the AA. These models are composed in the component called *Simulation server* that, in terms of ASML, is an *active entity.*

In the considered fragment of the ATC system, the set of use cases is developed for individual aircraft, since all operate equally. The following use cases are considered below: *ATC in arrival zone*, corresponding to elaboration (computation) of the landing plan by an aircraft; *Aircraft grouping*, intended for rough evaluation, by an individual aircraft, of a subset of other aircraft that potentially can be the sources of conflicts. The use cases involving the *Simulation server* include *Airliner initialization* when an aircraft enters the arrival zone, instead of *Simulation* when the current time variable is increased for one simulation duration. Each of the above use cases assumes interactions according to the corresponding protocols.

Role discovery results in one role, *Pilot (aircraft) assistant* (PA). It should participate in these use cases to be responsible for autonomous planning and scheduling of landing trajectory, prediction of potential conflicts with other aircrafts and conflict

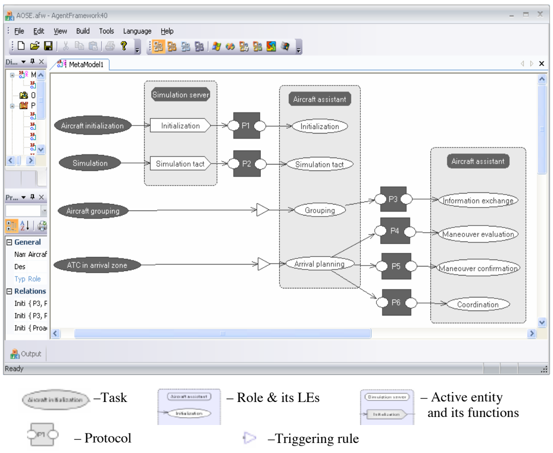


Fig. 4. Example of MAS macro model diagram

resolution through negotiation and reaching agreement. Accordingly, the listed use cases determine the list of LEs of the PA role while determining its interaction model and list of protocols. The single agent class, *PA-agent class* assigned the PA role is introduced.

An important component of the macro model is a set of *organizational rules*. In this case study, organizational rules represent the distributed safety policy, but since analysis of this is not the focus of this paper, its description is omitted.

MASDK 4.0, implementing ASML, supports all aforementioned analysis-related activities with graphical means. Fig. 4 shows the MADK 4.0 product, i.e. graphical notation of the *macro model* of agent-based autonomous ATC system and its components, representing PA role scheme and environment model, *Simulation server*. It is worth noting that this macro model is not only "a picture", but is also the formal model specified in ASML, and used as input for the next development stages. In general, architectural design should result in selection of the organizational structure, and refinement of the roles and interaction models. Below, these Gaia activities and their support in MASDK 4.0, as well as the products, are considered by example.

Organizational structure. The autonomous ATC assumes negotiations of PA roles in several use cases with no mediation by a centralized server, i.e. with no hierarchy. This requires use of P2P negotiations of the PA roles, so that agents have to negotiate using the distributed P2P platform that is a component of MASDK 4.0 environment [15].

Interaction model architectural design. Fig. 5 depicts an example of the protocol specification. It represents the protocol P4 identified in the macro model (Fig. 4). According to the latter, this protocol is used by a PA role when it agrees with maneuvers proposed by the same role of other PA agent class instances. In the diagram, these roles are denoted as *I-Airliner* (the protocol initiator) and *P-Airliner* (participant, or respondent of the P4 protocol) respectively. In the example, only one respondent of the P4 protocol

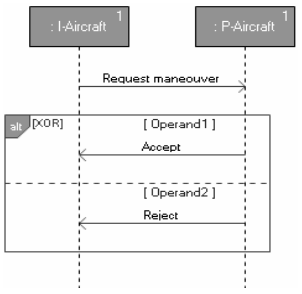


Fig. 5. Example of a protocol diagram

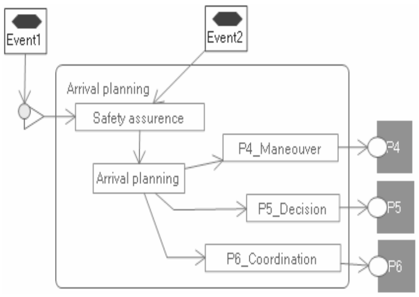


Fig. 6. Example of a scenario diagram

exists. It receives a request from *I-Airliner* to perform some maneuver, either accepting the request or rejecting it, while sending a corresponding message, either *Accept or Reject* respectively (Fig. 5). In Fig. 5 the protocol description consists of two participants and includes one combined fragment. In general, the protocol can consist of several participants and can include several nested combined fragments. It is worth noting that UML *Sequence diagrams* are used as prototypes of the protocol diagram used in MASDK 4.0. The limitation of such diagrams is that they do not allow for specifying nested protocols so far. This limitation is actually made up for by other types of diagrams aimed at specification of the role scheme.

Thus, the *second important product* of MASDK 4.0 in architectural design is the set of formal protocol models composing the MAS interaction model that is specified in terms of UML-like Sequence diagrams. Let us note that this specification is later used in automatic generation of the corresponding LE scenario scheme represented via the set of its nodes and transitions between them.

Role model: Liveness expression specification. Role scheme diagram. Architectural design of an agent role consists in specification of its LEs. It is done using two types of diagrams, *Role scheme* and *Behaviour scenario*. The former diagrams are used for decomposition of the LE into several simpler behaviour sub-scenarios, as well as for specification of the inter-roles behaviour coordination scheme.

Fig. 6 explains by example the decomposition rules described above. It represents the PA *Role scheme* diagram for the LE *Arrival planning* (Fig. 4). According to the macro model, this LE is capable of proactive behaviour, and participates in three interaction protocols, $P4$, $P5$ and $P6$. Accordingly, this LE scheme is decomposed to four simple behaviour scenarios denoted as *Safety assurance, P4_Maneuver, P5_Decision* and *P6_Coordination*. The fifth simple scenario, *Arrival planning*, is added to provide a specification of the logically complete sub-scenario that is the computation of the admissible set of landing plans. The arrows connecting the scenarios are interpreted as *"scenario – invokes – scenario"* imposing order on performance of simple scenarios.

E.g., according to the diagram given in Fig. 6, the simple scenario *Safety assurance* is initiated by a proactive behaviour rule. This rule is triggered when *event 1* has happened, for instance. The latter is generated as an output of another LE, *Simulation* (Fig. 4), when an airliner is approaching the airport airspace point connected to holding area. In

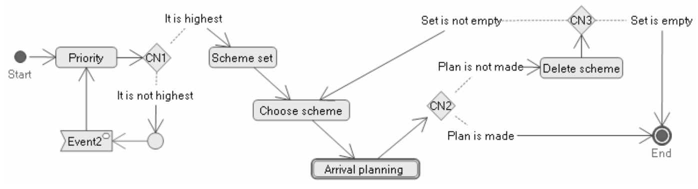


Fig. 7. Example of scenario diagram

that point, the PA role has to compute a flight plan for moving within the next sector of the arrival scheme. If, at the same time, other PA-agent instances begin computing their flight plans within the same sector then, according to the distributed safety policy, these PA-agent instances have to compute their priorities using the *Safety assurance* scenario. These priorities determine the order of their entry in the same sector, while granting entry permission to the highest priority aircraft. If an aircraft is not granted this permission, it interrupts execution of the *Safety assurance* scenario and waits for *event 2*, which arrives when the situation is changed in a way leading to a change of the aircraft priorities. In this case, the *PA-agent* instance repeats computation according to the above described scenario and, in case it has highest priority, continues performance of the LE and invokes the *Arrival planning* scenario, determining its behaviour within the next sector.

Role model: Liveness expression specification. Behaviour scenario diagram. Behaviour scenario diagrams are used for specification of simple scenarios. This type of diagram is an extension of UML activity charts [19]. Two types of simple scenarios are discerned, associated and not associated with the interaction protocols, and specified differently. Fig. 7 presents a self-explanatory example of a scenario that is not associated with any interaction protocol.

Behaviour scenarios of the second type, i.e. associated with participation of the role in an interaction protocol, must be consistent with the protocol specification, (protocol specification determines scenario behaviour logic). In the MASDK 4.0 environment, this dependency is automatically supported by two mechanisms. The first provides for automatic generation of scenario behaviour logic, and the second allows for preservation of this logic during further development (refinement) of the scenario by the designer. Both these mechanisms are explained in Fig. 8 where description of the *P4_Maneuver* scenario is shown. This scenario specifies the performance of the PA role (as initiator) in interaction protocol P4 (Fig. 4) represented by the diagram in Fig. 5.

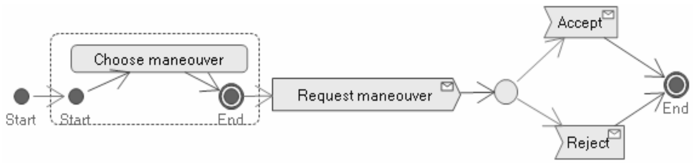


Fig. 8. Diagram example of scenario related to protocol

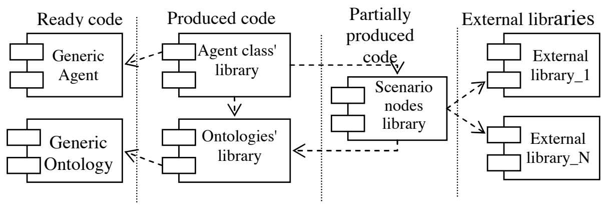


Fig. 9. Class libraries' scheme

The core of the detailed design is in depth development of the agent classes and service models. Let us outline how these activities are supported by MASDK. Agent classes inherit behaviours of the roles assigned. In particular, they inherit the roles' schemes and behaviour scenarios that require detailed specification of the agent class model (scenario nodes and transitions between them) in terms of variables and attributes.

This task of detailed design is aimed at more precise specification of agent class behaviour in terms of the agent class model variables, attributes, behaviour scenarios and functionalities of the scenario nodes. Variable and attribute types are defined either according to the ontology concepts used, or by standard data types usually supported by programming languages.

The resulting specification is used as an input to the SCB (Fig. 1), which checks MAS model specification correctness and either reports (through messaging) on the developed model incompleteness or inconsistency, or automatically builds source code of agent classes. In the former case, the messages inform the designer about what has to be redesigned in the model specification, e.g. the messages can be as follows: *"Description of behaviour scenario ⟨name⟩ is not completed"*, or *"Proactive behaviour of liveness expression ⟨name⟩ is not described"*. Model completeness and consistency can also be checked during the design process, i.e. independently of generation of the source code. This helps the designer to earlier fix the current state of MAS model development.

The architecture of the agent class software produced by SCB is depicted in Fig. 9. *Generic agent* and *Generic ontology* are provided by MASDK 4.0 environment as ready (reusable) components in terms of source code from the very beginning. They represent invariant behaviour of agent classes and abstract ontology. *Generic agent*, in particular, includes classes representing the abstract behaviour scheme of agent classes, the abstract proactive behaviour model of agent classes, abstract simple scenarios, etc.

The source code of *Agent class* and *Ontology* components is generated by the SCB component. For this purpose, the SCB translates elements of the model specification of each agent class to the corresponding class libraries. In particular, simple behaviour scenarios of an agent class and scenario performance order are represented by the *Agent class library.* The *Ontology library* provides for access to data and knowledge storage of *Agent class*. The source code of these two libraries is automatically generated in full without any additional programmer efforts.

The *Scenario nodes library* specifies scenario node functionalities, whose classes and methods correspond to simple scenarios of agent class and their nodes, respectively. Specification of scenario nodes includes identification of their names, and informal description (comment) and strict specification of their input/output attributes. Therefore, source code generated by SCB component is composed of the headers, the variables and the attributes of the classes and their methods, but the “bodies” of methods must be encoded by programmers manually. The agent class can use some external entities (e.g., applications, resources, etc). If they are not agent-based software, these components are developed externally, outside the MASDK 4.0 environment.

Development of MAS can be of iterative nature. In this case, the main problem is the maintenance of the source code developed earlier. This problem is solved by the SCB, which reports which classes and methods 1) are new, 2) which of them can be reused since they were developed earlier and do not require any modification, 3) which of them have to be rewritten according to MAS model modification, and 4) which of them should be deleted. It is worth noting that the source code maintenance problem concerns only the *Scenario nodes* component. The components *Agent class* and *Ontology* are generated by the SCB anew using the new MAS model specification.

6 Related Work

Since the first publication [20], Gaia was accepted as a valuable abstract methodology focusing on organizational issues. Many publications were devoted to its extensions and in-depth development to make it practically applicable.

Cernuzzi et al [3] proposed enriching Gaia with AUML for protocol specification. Indeed, the interaction model is a key issue of the organizational model, and adding the formal notation of AUML to Gaia can significantly enrich the methodology. The Agent Interaction Protocol (AIP) of AUML regards the protocol as an integrated entity combining roles, constraints, and communication acts, while clearly expressing, in UML-like notation, the formal protocol semantics. The paper pays specific attention to the problem of modeling the complexity of open MAS and emergent behaviours. It introduces a distinguished set of agent class instances allocated the same roles while Gaia just specifies the role. An additional argument for AUML is that it is capable of specifying timely message ordering that is important to implement concurrency.

Garcia-Ojeda, et al [10] propose a similar extension of Gaia, using benefits provided by AIP of AUML. In fact, UML is well developed and widely used as a de-facto software design standard and, therefore, would be accepted by practitioners. The authors proposed to refine the architectural design interaction model based on the two first layers of AIP in terms of AUML for more detailed interaction model representation. Role and service models (at architectural and detailed design) are refined via integration of the AIP third layer and extended UML Class Diagrams. Finally, they refine the organizational model by integrating all the Gaia models developed at previous stages using the Alaadin model proposed by Ferber et al [7], that naturally provides a basis for developing representational mechanisms for organizational concepts.

In order to make Gaia more practically applicable, Gonzalez-Palacios, et al [13] proposed an extension of Gaia in two aspects. First, they consider the design of the internal composition of agents that is omitted in Gaia. This activity uses, as the input, the organizational design results. The proposed agent model is composed of two parts, the *structure model* and the *functionality* model. The first model decomposes roles into classes, while the functionality model is intended to specify collaboration of the above classes determining the expected role classes behaviour. An advantage of this approach is that such design activity is independent of the specific agent architecture (reactive, BDI, etc.). The second extension regards an iterative approach to large scale application development. The whole development process is divided into simply manageable units that are analyzed, designed and implemented one after another, thus extending previously produced executable deliverables. The final deliverables must contain all the functionality expected from the system.

The Roadmap methodology described in [14] is motivated by the desire to extend Gaia to engineering of large scale open systems, viewing the latter as computational organizations. Roadmap introduces use cases in order to discover requirements (like MASE [6]), make explicit agent environment and knowledge models, and also to enrich Gaia by interaction AUML-based models. However, the authors do not regard these refinements in the needed detail, while other issues remain unclear.

The main drawback of the reviewed and other existing works is that all of them deal mostly with particular stages or aspects of Gaia, extending its particular modeling issues. In contrast, in this paper we treat the whole lifecycle of Gaia, while proposing a formal specification language supported by a powerful software tool, MASDK 4.0, implementing a model- driven agent-based software engineering approach.

7 Conclusion

The MAS development environment, MASDK 4.0, thoroughly implements the proposed extension of the Gaia methodology. It supports user-friendly technology for MAS application development. Its advantages and novelties are as follows:

1. It is based on an extended version of the well founded *Gaia methodology*. The proposed *extensions* are intended for making Gaia applicable to practical use.
2. It realizes a *model-driven engineering approach*, providing automatic support for consistency and integrity of all the intermediate and final solutions produced by developers at all Gaia development stages. The use of model-driven engineering allows for substantial speeding up of the development process. This approach is supported by the *formal specification language, ASML,* that is provided with graphical notation, thus supporting the user-friendly graphical design style.
3. MASDK 4.0 *supports* the development activity *at all development stages*: at the *analysis stage* when organizational issues of the MAS macro-model are decided; at the architectural and detailed *design stages* where the development process is represented in graphical notation of the formal specification language; and at semi-automatic *code generation and deployment stages.*
4. MASDK 4.0 provides *automatic generation of agent behaviour scenarios* schemes using formal protocol specifications as input. This is one of the most important advantages of MASDK4.0, distinguishing it from other existing MAS development environments and tools.
5. MASDK 4.0 is integrated with a *FIPA-compatible distributed P2P agent platform*, supporting a service-oriented approach. It can provide a distributed P2P infrastructure for interaction of heterogeneous software agents running over various operating systems and using heterogeneous communication protocols.

The MASDK environment was tested on various applications during recent years. Its current runtime version and documentation are available on the web [16]. In addition, the FIPA compliant agent platform (runtime version) is freely available. This version, together with the documentation, can be found at [15].

Future work is aimed at: thorough testing of the current version on various applications, including embedded and mobile ones in order to determine the directions of Gaia and MASDK's further enrichment to make it of industrial level. The intended enrichment of the FIPA compliant P2P agent platform should also be done in order to provide it with more system services and capabilities supporting operation of heterogeneous nomadic agents. Development of the light versions of the MASDK environment specifically intended for mobile applications is also planned.

References

1. AgentLink. Agent Software, `http://eprints.agentlink.org/view/type/software.html`
2. Bellifemine, F., Poggi, A., Rimassa, G.: JADE — A FIPA-compliant agent framework, `http://sharon.cselt.it/projects/jade/papers/PAAM.pdf`
3. Cernuzzi, L., Zambonelli, F.: Experiencing AUML in the Gaia Methodology. In: 6th International Conference on Enterprise Information Systems — ICEIS 2004, Porto, Portugal (2004)
4. Coguaar web site, `http://www.cougaar.org`
5. DeLoach, S., Wood, M.: Developing Multiagent Systems with agentTool. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS, vol. 1986, p. 46. Springer, Heidelberg (2001)
6. DeLoach, S., Wood, M., Sparkman, C.H.: Multiagent systems engineering. International Journal of Software Engineering and Knowledge Engineering 11(3), 231–258 (2001)
7. Ferber, J., Gutknecht, O.: A Meta-Model for the Analysis and Design of Organizations in Multiagent Systems. In: Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS 1998), pp. 128–135. IEEE Computer Society, Los Alamitos (1998)
8. FIPA P2P NA WG6, Functional Architecture Specification Draft 0.12, `http://www.fipa.org/subgroups/P2PNA-WG-docs/P2PNA-Spec-Draft0.12.doc`
9. FIPA ACL Message Structure Specification, `http://www.fipa.org/specs/fipa00061/SC00061G.htm`
10. Garcia-Ojeda, J., Arenas, A., Perez-Alcazar, J.: Paving the way for implementing multiagent systems. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 179–189. Springer, Heidelberg (2006)
11. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos software development methodology: Processes, Models and Diagrams. In: Giunchiglia, F., Odell, J.J., Weiss, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 162–173. Springer, Heidelberg (2003)
12. Gomez, J., Fuentes, R., Pavon, J.: The INGENIAS Methodology and Tools. In: Agent-oriented Methodologies, pp. 236–275. Idea Publishing Group, USA (2005)
13. Gonzalez-Palacios, J., Luck, M.: Extending Gaia with Agent Design and Iterative Development. In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 16–30. Springer, Heidelberg (2008)
14. Juan, T., Pearce, A., Sterling, L.: ROADMAP: Extending the Gaia Methodology for Complex Open Systems. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), pp. 3–10. ACM, New York (2002)
15. LIS Agent Platform, `http://space.iias.spb.su/ap/`

16. MASDK 4.0, `http://space.iias.spb.su/masdk`
17. Reticular Systems Inc.: AgentBuilder An Integrated Toolkit for Constructing Intelligent Software Agents. Revision 1.3 (1999), `http://www.agentbuilder.com/`
18. Rimassa, G., Kernland, M., Ghizzioli, R.: LS/ABPM — An Agent-powered Suite for Goal-oriented Autonomic BPM. In: Proceediongs of AAMAS 2008, Portugal (2008)
19. Unified Modeling Language: Superstructure, `http://www.omg.org/docs/formal/07-02-05.pdf`
20. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing Multiagent Systems: The Gaia Methodology. Transactions on Software Engineering and Methodology 2(3), 317–370 (2003)

Author Index