AAMAS 2010
TORONTO

The 9th International Conference on
Autonomous Agents and Multiagent Systems
May 10-14, 2010
Toronto, Canada

Workshop 8

The Eleventh International Workshop on Agent Oriented Software Engineering

AOSE 2010

Editors:
Wiebe van der Hoek
Gal A. Kaminka
Yves Lespérance
Michael Luck
Sandip Sen

THE ELEVENTH INTERNATIONAL WORKSHOP ON AGENT-ORIENTED SOFTWARE ENGINEERING

Toronto, Canada
10 of May 2010

Preface

Since the mid 1980s, software agents and multi-agent systems have grown into a very active area of research and also commercial development activity. One of the limiting factors in industry take up of agent technology, however, is the lack of adequate software engineering support, and knowledge in this area.

AOSE is focused on this problem and provides a forum for those who study the synergies between software engineering and agent research.

The concept of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a natural one for software designers. Just as we can understand many systems as being composed of essentially passive objects, which have state, and upon which we can perform operations, so we can understand many others as being made up of interacting, autonomous or semi-autonomous agents. This paradigm is especially suited to complex systems.

Software architectures that contain many dynamically interacting components, each with their own thread of control, and engaging in complex coordination protocols, are typically orders of magnitude more complex to correctly and efficiently engineer than those that simply compute a function of some input through a single thread of control, or through a limited set of strictly synchronized threads of control. Agent oriented modelling techniques are especially useful in such applications.

Many current and emerging real-world applications - spanning scenarios as diverse as worldwide computing, network enterprises, ubiquitous computing, sensor networks, just to mention a few examples – have exactly the above characteristics. As a consequence, agent oriented software engineering has become an important area: both as a design modelling means, and as an interface to platforms which include specialised infrastructure support for programming in terms of semi-autonomous interacting processes.

The particular focus of AOSE 2010 will be on how to bridge the gap between AOSE and conventional software engineering. We aim to look at the integration of concepts and techniques from multi-agent systems with conventional engineering approaches on the one hand, and the integration of agent-oriented software engineering and methodologies with conventional engineering processes on the other hand.

Marie-Pierre Gleizes
Danny Weyns
(Editors)

A workshop of AAMAS 2010

The Eleventh International Workshop on Agent-Oriented Software Engineering

Workshop Chairs

Marie-Pierre Gleizes	IRIT, Universitè Paul Sabatier, Toulouse, France
Danny Weyns	DistriNet Labs, Katholieke Universiteit Leuven, Belgium

Steering Committee

Programme Committee

Aditya Ghose (University of Wollongong, Australia)
Adriana Giret (Technical University of Valencia, Spain)
Alessandro Garcia (PUC Rio, Brazil) Alessandro Ricci (Universita di Bologna, Italy)
Anna Perini (Fondazione Bruno Kessler, Italy)
Brian Henderson-Sellers (University of Technology, Australia)
Carole Bernon (University Paul Sabatier, France)
Eric Yu (University of Toronto, Canada)
Fariba Sadri (Imperial College, UK)
Flavio Oquendo (Universit de Bretagne-Sud, France)
Frdric Migeon (Paul Sabatier University, France)
Gauthier Picard (ENS Mines Saint-Etienne France)
H. Van Dyke Parunak (TechTeam Government Solutions, USA)
Haralambos Mouratidis (University of East London, UK)
Holger Giese (Hasso Plattner Institute Postdam, Germany)
Jeffrey Kephart (IBM T.J. Watson Research Center, USA)
Joao Leite (Universidade Nova de Lisboa, Portugal)
Jorge J. Gmez Sanz (Universidad Complutense de Madrid, Spain)
Juan Antonio Botia Blaya (Universidad de Murcai, Spain)
Juergen Lind (Iteratec, Germany)
Laszlo Gulyas (AITIA International Inc.,Hungary)
Mark Klein (Software Engineering Institute, Carnegie Mellon, USA)
Massimo Cossentino (ICAR-CNR, Italy)
Michael Winikoff (University of Otago, New Zealand)
Michael Zapf (Universitt Kassel, Germany)
Michal Pechoucek (Czech Technical University in Prague, Czech Republic)
Onn Shehory (Haifa University, Israel)
Paolo Giorgini (University of Trento, Italy)
Philippe Mathieu (University of Lille, France)
Ruben Fuentes (Universidad Complutense, Spain)
Scott DeLoach (Kansas State University, USA)
Simon Miles (King's College London, UK)
Tom Holvoet (K.U. Leuven, Belgium)
Valeria Seidita (University of Palermo, Italy)
Vicente Julian Inglada (Universidad Politecnica de Valencia, Spain)
Vincent Hilaire (Belfort-Montbeliard Technology University, France)
Virginia Dignum Delft University of Technology, The Netherlands)
Viviana Mascardi (Universit di Genova, Italy)
Yuriy Brun (University of Washington, USA)

Table of Contents

A Value-Sensitive Approach to Agent-Oriented Software Engineering

Christian Detweiler, Koen Hindriks, Catholijn Jonker

{c.a.detweiler,k.v.hindriks,c.m.jonker}@tudelft.nl

Abstract. Prominent agent-oriented software engineering methodologies such as Tropos support the engineer throughout most of the development process. Though in this method attention is paid to system stakeholders by explicitly modeling them, potential harms and benefits of the system to these stakeholders, and the underlying human values that are impacted, are not explicitly accounted for. Value-Sensitive Design is a methodology that does address such issues, but offers little guidance in operationalizing them. We demonstrate differences between these methodologies using the development of a Conference Management System as a case study. Subsequently, we propose a means of integrating the methodologies to operationalize Value-Sensitive Design and take values into account in agent-oriented software engineering.

1 Introduction

In making choices in the design of information systems, designers "necessarily impart social and moral values" [3]. Beyond such choices, once a system has been put into use it affects its direct stakeholders and indirect stakeholders and their values. Privacy issues with social networking websites, bias in search engines, and intellectual property issues with file-sharing software are a few examples of value issues that are brought about by technology and that originate in their design.

Few design methodologies explicitly take values into account in the design process. Values are treated informally and left implicit, leaving them in essence unincorporated in the design.

Value-Sensitive Design [5] (henceforth VSD) provides a comprehensive framework for making value issues explicit in designing a system, but provides little to concretize these values. Agent-oriented software engineering (AOSE) methodologies such as Tropos do focus on stakeholders' goals, but do not explicitly take values into account. Human values are left implicit as goals, and as such are lost in the details of the design.

This paper is organized as follows. In section 2 we briefly discuss VSD and one of the more well-known and comprehensive AOSE methodologies, Tropos. Then, we discuss a case study to identify issues related to values that are not explicitly supported by current agent-oriented software engineering methodologies in section 3. In section 4, we analyze important differences between central concepts in VSD and Tropos. In section 5, we present our proposal for an integration of VSD into AOSE, and conclude this paper in section 6.

2 Background

2.1 Value-Sensitive Design

VSD is an approach to the design of technology that accounts for human values the design process [4]. In VSD, emphasis is given to supporting moral values or values with ethical import, such as human welfare, ownership of property, privacy, and freedom from bias [5].

VSD provides an iterative and integrative three-part methodology consisting of conceptual, empirical, and technical investigations. Conceptual investigations focus on discovering affected stakeholders, their values, and analyzing these values and tensions between them [7]. The designer identifies direct and indirect stakeholders and potential harms and benefits to those stakeholders. He or she then maps these onto associated values, especially human values with ethical import. Next, the designer clearly defines these values. Potential value conflicts are then examined.

Conceptual investigations need to be informed by empirical investigations of the technology's context, in which "the entire range of quantitative and qualitative methods used in social science research is potentially applicable" [5].

Technical investigations can either focus on the properties and mechanisms of existing technologies that support or hinder human values, or can consist of designing a system to support identified human values.

It could be argued that the steps taken in VSD are common sense. Common sense as it may be, values are often neglected in design and addressed after the fact, as cases of privacy issues with social networking websites, bias in search engines, and intellectual property issues with file-sharing software illustrate.

2.2 Tropos

The Tropos software development methodology supports the agent-oriented paradigm and the associated concepts of actors, plans and goals throughout the software development process [1-2, 6]. The emphasis placed on stakeholders in Tropos makes it closely related to, and therefore suitable for combination with, VSD.

Tropos identifies stakeholders early in the design process, in the Early Requirements phase. Stakeholders' goals are identified next, and for every goal the developer decides whether the actor itself can achieve it or it needs to be delegated to another actor. Actors, and goal dependencies between them, are captured in an Actor Diagram.

Actors' goals are then decomposed (through AND/OR, means-end, or contribution analysis) into sub-goals and plans. Non-functional requirements, represented as soft-goals, can also be identified at this point. The goal-decomposition process results in a Goal Diagram.

The designer introduces the system-to-be as an actor in the Late Requirements phase and delegates goals to it. The activities in this phase result in an extended domain model and the Late Requirements Actor and Goal diagrams.

The Architectural Design phase focuses on defining the global architecture of the system, and the Detailed Design phase further refines these models to make them operational.

3 Case study

3.1 Value-Sensitive Design

To apply VSD to the Conference Management System case, we conducted semi-structured interviews with stakeholders to elicit potential harms and benefits, and underlying values. We elicited these based on the context of use of the future conference management system.

Several stakeholders mentioned potential harms related to the anonymity of reviewers. They stated that anonymity removes context, making it difficult to assess reviewers' expertise and damaging the quality of discussion. Also, it allows reviewers to "ride their hobby horse", posing a threat to their objectivity. Transparency and accountability are important underlying values here.

Several interviewees also expressed concerns regarding the possibility of conflicts of interest introduced by users being able to occupy multiple roles within the same system. For example, PC members that are also authors could see the ranking of their submission, or reviewers could review papers of colleagues.

The most frequently mentioned benefits were related to anonymity. Several interviewees stated that anonymity removes hierarchical considerations, leading to judgments based on quality and not on academic position. It also allows reviewers to be as critical as (they feel) they need to be. Equality and privacy are underlying values here.

Many saw the possibility of using the same CMS for multiple conferences as a potential benefit, as it enhances the trustworthiness of the system and the process it supports. Also, the record of interactions with the system that could be provided enhances the transparency and accountability.

3.2 Tropos

We will focus on the Analysis phase of Tropos in this case study, as the level of abstraction in this phase is most closely related to that of VSD as described above. The case study described in [2], begins with the identification of stakeholders, represented as the actors **Author**, **PC**, **PC Chair**, **Reviewer**, and **Publisher**.

The following step is goal identification. For example, there is a goal dependency between **Author** and **PC** for the **Peer review** goal. The designer then decomposes goals. For example, the **Manage Conference** goal is AND-decomposed into the (satisfiable) sub-goals **Get papers**, **Select papers**, **Print proceedings**, **Nominate PC** and **Decide deadlines**. Satisficeable soft-goals are also analyzed, such as the **PC Chair** actor's **Conference quality**, to which the soft-goal **better quality papers** contributes positively [2].

Next, the designer introduces the system-to-be as an actor to which (human) actors' goals can then be delegated. For example, the PC delegates the **Coordinate conference** goal to the CMS System actor, which is then further decomposed into sub-goals and eventually plans. Design is then continued in the Architectural Design and Detailed Design phases of Tropos, which will not be discussed here.

4 Analysis

The case studies above illustrate a number of a number of similarities and differences between VSD and Tropos.

The central role of stakeholders is a strength of both methodologies. Both begin their analyses by identifying stakeholders. After stakeholder identification, VSD proceeds to identify stakeholders' values, and Tropos identifies their goals.

Both methodologies examine an existing practice in which a system will be introduced, through focus on the context of use and actor networks into which a system is introduced, respectively.

Another overlap seems to exist between VSD's values and non-functional requirements captured in Tropos' soft-goals. However, there are some important differences to consider.

The implicit aim of VSD is to design systems that are good and do no wrong in an ethical sense. To this end, VSD calls for uncovering potential harms and benefits and the values they affect. Potential harms are to be avoided, so that underlying values are not hindered. Potential benefits are to be promoted, so that underlying values are supported.

So, in VSD, values lead to norms for supporting and not hindering those values. For example, upholding the value of privacy generally means following norms of distribution of (personal) information. Violating such norms constitutes a violation of privacy and is morally wrong.

In this sense values are not merely non-functional requirements that can be captured in Tropos' soft-goals. We cannot say that not satisficing a soft-goal as such is morally wrong. We can say that hindering a moral value is. Also, soft-goals in themselves do not lead to norms in the way values do.

If we were to consider values non-functional requirements and represent them as soft-goals in Tropos, there would be no explicit distinction between values, such as privacy, and other non-functional requirements, such as conference quality. Representing values as soft-goals would leave them implicit.

A strength of VSD is that it makes values explicit, so that they are a focal point throughout design. A weakness of VSD is that values are left abstract. Values are not concretized and connected to actual designs, making it difficult to assess specifically how they are incorporated into designs and how values supported (or hindered) by the design.

5 Value-Sensitive Agent-Oriented Software Engineering

To design systems that perform well in a moral sense as well as in a functional sense, we need a way of making values explicit and concrete. As discussed above, VSD helps to make values explicit, but does not make them concrete. We have argued that soft-goals in Tropos are not sufficient to capture (and thus concretize) values, because values cannot be considered non-functional requirements as such. Therefore, we propose a number of extensions to the Tropos methodology.

The first of these is a set of "design questions" that help designers using Tropos to identify values and distinguish them from "normal" goals. After identifying stakeholders, their goals, their values, and potential harms and benefits to stakeholders, the designer can ask the following questions to assess whether the elicited item is a value or a goal. These questions are derived from the analysis in section 4. *Can not meeting the requirement be considered wrong in a moral sense? Can meeting the requirement be considered morally good? Does this requirement lead to norms that can be violated or conformed to?*

In this case, not achieving the requirement of privacy (e.g. of reviewers) can be considered morally wrong. It leads to norms of distribution of information, in this case. Violating such norms could be considered morally wrong.

Now that values can be identified and distinguished from goals, they must be represented in Tropos models to be able to properly take them into account. As discussed above, values cannot be captured directly in Tropos goals or soft goals.

So, we propose introducing new model entities to Tropos models: the value and the actor-value link. As argued, values cannot be captured in goals or soft-goals, and as such should be entities in themselves. Values are elicited from stakeholders, so should be attached to stakeholders in models. Values as such are not delegated to other actors as goals are in Tropos, so a means is required to connect actors and values without the need to create dependencies between actors. The actor-value link fulfills this function.

As discussed above, values lead to norms that can be violated or conformed to. Values should then be linked to such norms, to make explicit how the value is operationalized and what the source of a norm is. Some work has dealt with norm specification in Tropos (e.g. [8]). However discussion of norm specification in Tropos is beyond the scope of this work.

Consider, for example, the value of privacy elicited from reviewer stakeholders. This value would be represented as a value entity in Tropos and linked, by means of an actor-value link, to the **Reviewer** actor. It leads to a norm of sharing personal information appropriately. In this case, the designer would introduce the norm **Do not share personal information** for (indirect) dependencies between the **Reviewer** and the **Author** actors. This could be operationalized as a **Shield identity** goal that would be an AND-decomposed sub-goal of the **PC** actor's goal **Collect reviews**.

6 Conclusions

We have argued that human values are impacted by technological designs. However, few methodologies explicitly take such values into account. We described VSD, a methodology that aims to account for (moral) values in design. We argued that VSD is a useful methodology for eliciting stakeholders' values and making them explicit.

However, VSD as-is leaves values abstract. It does not provide a means for concretizing such values. This makes it difficult to assess the extent to which values are incorporated in actual designs.

Tropos seems well-equipped to deal with stakeholders' values due to their focus on stakeholders and their goals. However, we argued that Tropos' concept of soft-goals, which can be used to represent non-functional requirements, is fundamentally different from human values as described here. Representing values as soft-goals does not make values sufficiently explicit as values.

To address these problems, we proposed complementing Tropos with two model entities: the value and the actor-value link. These entities will allow the designer to explicitly represent values throughout the design process, and to make values concrete enough to operationalize them.

References

1. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. and Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8(3):203-236, (2004)
2. DeLoach, S., Padgham, L., Perini, A., Susi, A. and Thangarajah, J.: Using three AOSE toolkits to develop a sample design. International Journal of Agent-Oriented Software Engineering, 3(4):416-476, (2009)
3. Friedman, B. (ed.): Human Values and the Design of Computer Technology. Cambridge University Press and CSLI, New York, NY and Stanford, CA (1997)
4. Friedman, B., Kahn, P. and Borning, A.: Value sensitive design: Theory and methods. University of Washington Technical Report, (2002)
5. Friedman, B., Kahn, P. and Borning, A.: Value sensitive design and information systems. Human-Computer Interaction and Management Information Systems: Foundations. ME Sharpe, New York:348-372, (2006)
6. Giunchiglia, F., Mylopoulos, J. and Perini, A.: The tropos software development methodology: processes, models and diagrams. Lecture Notes in Computer Science:162-173, (2003)
7. Miller, J., Friedman, B. and Jancke, G.: Value tensions in design: the value sensitive design, development, and appropriation of a corporation's groupware system. *In Proc.* pp. 281-290. ACM (2007)
8. Siena, A.: Engineering Normative Requirements. *In Proc. 1st International Conference on Research Challenges in Information Science (RCIS'07)*. pp. 439–444 (2007)

A Semiotic Approach for Multiagent Systems Situational Development

Sara Casare[1], Anarosa A. F. Brandão[1], Jaime S. Sichman[1]
[1] Intelligent Techniques Laboratory – University of São Paulo - Brazil
{sara.casare, anarosa.brandao, jaime.sichman}@poli.usp.br

Abstract. Based on a semiotic approach, this paper proposes a MAS Semiotic Taxonomy that is a first step towards combining the advantages from AOSE methods and AI inspired AO models, in order to provide a set of categories that glues together both AOSE typical aspects and AO models characteristics, allowing a better identification of different MAS development aspects related to them. Moreover, it may serve as the basis for building MAS method fragments´ repositories that could be used to create MAS situational methods. The taxonomy has three complementary inspiration sources: (i) MAS specific development aspects originated from AOSE methods and AO models; (ii) Situational Method Engineering related concepts; and (iii) method content and method process notions.

Keywords: Software engineering (multiagent oriented), Social and organizational structure, Semiotic.

1 Introduction

In order to structure the development and to manage the complexity associated with Multiagent Systems (MAS), several development methods have been proposed in AOSE field during the last decade, e.g. Gaia [19], Tropos [1], MaSE [18], O-MaSE [8], Ingenias [15] and PASSI [3]. Additionally, following the AI tradition in MAS, different agent organization (AO) models such as AGR (*Agent, Group, Role*) [7], OperA (*Organization per Agent*) [6], and MOISE+ (*Model of Organization for Multiagent Systems*) [11] were proposed.

While a great part of methods adopts an agent centered MAS approach, focusing on the agent's behavior, agent organization models consider the notion of group or agent organization as a leading concept, called by some authors as an organization centered approach [13].

Since some AO models are not currently incorporated into AOSE based MAS development methods, someone who adopts an organization centered approach to build a MAS may not have tool support for using both AOSE methods and AO models together. Nevertheless, using one of them separately may cause some project drawbacks. On the one hand, MAS methods that offer a structured development cycle may not adopt an explicit agent organization model. On the other hand, most AO

models do not provide a structured MAS development cycle in terms of phases, tasks and work products.

The variety of AOSE methods and AO models is due to the specific needs raised on MAS development and to the different approaches adopted by MAS developers. It shows that a method cannot be general enough in order to be applied to every MAS development project without some level of customization [9]. Nevertheless, it seems that reinventing a method for each new project situation wouldn't be a best practice, given that there are a great number of available methods for MAS development. This scenario suggests that Method Engineering techniques [2] and, particularly, Situational Method Engineering [10] seem to be promising approaches to be considered for MAS development. A situational method is a method tailored and tuned to a particular project situation that is configured in a formal and computer-assisted manner out of standardized and proven building blocks - called *Method Fragments* - stored in a data base.

Based on a semiotic approach [17], this paper proposes a MAS Semiotic Taxonomy that can be viewed as a first step towards combining the advantages from AOSE methods and AI inspired AO models, in order to provide a set of categories that glues together both AOSE typical aspects and AO models characteristics, allowing a better identification of different MAS development aspects related to them. Moreover, it may serve as the basis for building MAS method fragments´ repositories that could be used to create MAS situational methods.

This paper is organized in four sections. Section 2 presents MAS development issues and situational method engineering aspects for MAS and section 3 outlines the set of categories that composes the MAS Semiotic Taxonomy. Finally, section 4 presents a discussion about usages for the Semiotic Taxonomy.

2 MAS Development Issues and Situational Method Engineering

Lemaitre and Excelente [13] suggest that the two main MAS research field approaches are related to its main components: agent and group of agents. Agent approaches propose several formalisms for representing individual agent architecture and agent's "internal" knowledge, as beliefs, intentions and desires, among others. Group approaches adopt a sociological and organizational vision for modeling MAS, involving organizations, teams and inter-agent relationships notions.

An AO model offers a conceptual framework and syntax for designing organizational specifications that can be implemented on a traditional agent platform or using some organizational middleware. In order to participate in an AO, agents are supposed to previously know the organization main characteristics. In this way, they can play available organization roles, contribute to achieve global goals, participate in organization interactions and be aware of organization norms, as permissions, obligations and rights [5].

Several researchers are dealing with situational method engineering for MAS development. Among them, Cossentino et al [4] reported the experience of creating a new process involving PASSI methodology, Rougemaille et al [16] discussed the main aspects related to MAS fragment definition, and Garcia-Ojeda et al [8] proposed

an organization-based MAS engineering process framework (O-MaSE) that extends MaSE meta-model in order to support organization centered MAS development.

Situational Method Engineering [2] is the sub-area of Method Engineering that addresses the controlled, formal and computer-assisted construction of situational methods out of method fragments. Roughly speaking, building a situational method consists on reusing portions of existing methods taking into account a given project situation that encompasses, for example, notions of class of application (as traditional and pervasive computing) and project perspectives.

Along with the project situation and method fragments repositories, situational method building involves distinct reuse mechanisms, such as assembly-based [10] and method configuration mechanisms [12]. While an assembly-based mechanism begins the situational method building with assembling a set of disconnected method fragments in a bottom-up fashion, method configuration mechanism does it by taking a base method and modifying it to cover the project situation in a top-down fashion, eliminating, adding or exchanging additional fragments captured from another methods.

However, reusing available MAS development approaches to build MAS situational methods raise issues related to the construction of a method fragment repository and to the creation of a situational method using fragments. SPEM 2.0 [14] can be used as a common meta-model for the method fragment repository, offering concepts (as task, activity, work product, role, guidance, category) for a common definition across MAS development approaches. In addition, general reuse mechanisms for situational method building, such as assembly-based and method configuration mechanisms, can be customized suitably for the MAS context. Nevertheless, such mechanisms raise questions as: (i) How could the MAS project situation be described? (ii) How could MAS development approaches, as AOSE methods and AO models, be described? (iii) Finally, how could MAS project situation be matched with several aspects of MAS development approaches to build an appropriated method for the MAS project?

The Semiotic Taxonomy presented in this paper constitutes a first step towards solving these issues, since it offers a structured way to describe MAS development approaches. Moreover, in a MAS situational method framework this taxonomy could provide the categorization for method fragments to build their repositories and to take part on a MAS situational reuse mechanism. At the best of our knowledge, we are not aware of researches proposing a broad collection of categories for classifying and describing MAS aspects in order to support the customization of situational methods for MAS. Such categories are outlined in the next section.

3 MAS Semiotic Taxonomy

Semiotics deals with the syntactic (structure), semantics (meaning) and pragmatics (usage) aspects of signs. Ronald Stamper [17] proposed the Semiotic Ladder to treat information as signs: he has extended the traditional division of Semiotics – syntactic, semantic and pragmatic - by including three new signs aspects called *social* (social dimensions), *empirics* (statistics properties) and *physical* (hardware properties).

Based on such semiotic approach, we propose to use a MAS Semiotic Taxonomy to classify several aspects involved in MAS development approaches. By using this taxonomy, someone that has a MAS project to be developed may search for a good choice among the existing MAS development approaches, such as AOSE methods and AO models.

As shown in figure 1, the MAS Semiotic Taxonomy involves the following levels: *Social Level, Pragmatic Level, Semantic Level, Syntactic Level* and *Empirical Level*. Although there are applications related to agents' physical aspects (as pervasive and ubiquitous systems) the physical level is not taken into account in this taxonomy since such aspects are related to project information infrastructure that is treated as project situation aspect [10] and not as a method fragment aspect.

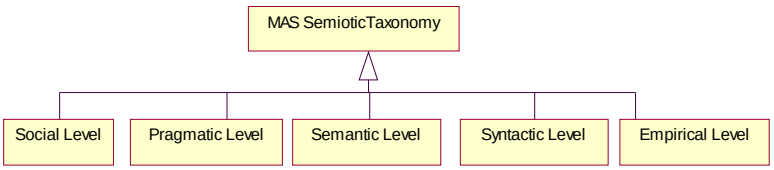


Fig. 1: Five levels of MAS Semiotic Taxonomy (as UML class diagram)

Using such a semiotic perspective, this taxonomy puts together concepts originated from three main sources: (i) MAS specific development aspects originated from AOSE and AO, (ii) Situational Method Engineering related concepts, mainly those proposed by Harmsen [10] and (iii) method content and method process notions provided by SPEM 2.0.

The *Social Level* aims to identify the set of social norms related to the MAS development aspects, involving the following categories: *Utilization Degree, Success Degree, Reuse Degree, Validation Degree, User Participation Degree, Iteration Type and Development Type*.

The *Pragmatic Level* allows distinguishing MAS development aspects based on their usage and intention. It is composed of the following categories: *Agent Discipline Category, Group Discipline Category, Analysis Style Category, MAS Approach Category, Fragment Source Category, MAS Nature Category and Agent Architecture Category*.

The *Semantic Level* allows distinguishing MAS development aspects based on their meaning, i.e., method fragment specific meaning into a software and process engineering meta-model as SPEM 2.0. Therefore, this level is mainly composed of method engineering typical aspects: *Fragment Content Category and Fragment Process Category.*

The *Syntactic Level* allows distinguishing MAS development aspects according to their structure and format. This level takes into account categories related to the notation and the language used to structure and to express them: *Fragment Notation Category, Language Paradigm Category and Fragment Language Category*.

Finally, the *Empirical Level* allows distinguishing MAS development aspects according to their development standards and patterns, involving the following categories: *Code Generation Category and Development Platform Category*.

For instance, a MAS project developer looking for components reusing (*Social Level - Reuse Degree Category*) could select method fragments sourced from MaSE, while the application of a goal based requirement analysis in the MAS project (*Pragmatic Level - Analysis Style Category*) as well as a graphical design notation (*Syntactical Level - Fragment Notation Category*) could consider method fragments sourced from Tropos. Additionally, the development of a MAS project that will follow an organization centered approach (*Pragmatic Level –MAS Approach category)* could take into account fragments captured from MOISE+ or AGR. Finally, if the MAS project developer considers a model driven approach (*Empirical Level - Code Generation Category)* method fragments could be sourced from Ingenias.

4 Conclusions

In this paper we have shown that a semiotic perspective could facilitate putting together MAS development approaches from both AOSE and AO fields in order to take advantages from all of them to build organization centered MAS.

We claimed that a semiotic approach can offer the backbone for describing MAS development aspects, given that it allows identifying several aspects typically involved in MAS development, answering part of the questions raised in section 2. Some of these aspects are related to its intention and usage (pragmatic), its meaning (semantic) and its structure (syntactic),

In a MAS situational method framework, this taxonomy could provide both: the categorization for method fragments to build their repository and to help on a MAS situational reuse mechanism. In an assembly-based mechanism scenario this taxonomy could be used to support the method fragment selection in a bottom-up fashion and, in a configuration mechanism scenario it could be used to support the MAS base method selection in a top-down fashion, as well as additional method fragments.

Acknowledgments. This work is part of the MEDEIA project from FAPESP, Brazil, grant 2009/10121-4. Authors are partially supported by CNPq and CAPES, Brazil.

References

[1] 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, vol 8(3), 203--236 (2004)

[2] Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools. Information and Software Technology, vol. 38 (4), 275--280 (1996)

[3] Cossentino, M.: From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, 79--106 (2005)

[4] Cossentino, M.; Galland, S. ; Gaglio, S.; Gaud, N.; Hilaire, V.; Koukam, A.;Seidita, V.: A MAS metamodel-driven approach to process composition. In: 9th International Workshop on Agent-Oriented Software Engineering (AOSE) (2008)

[5] Coutinho, L. ; Sichman, J. S. ; Boissier, O.: Modelling Dimensions for Agent Organizations. In: Virginia Dignum. (Org.). Multi-Agent Systems: Semantics and Dynamics of Organizational Models. Hershey: IGI Global, 18--50 (2008)

[6] Dignum, V. A model for organizational interaction: based on agents, founded in logic. Phd thesis, Universiteit Utrecht (2004).

[7] Ferber, J.; Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Agent-Oriented Software Engineering IV International Workshop (AOSE2003), LNCS, vol. 2935, Springer, 214–230 (2004)

[8] Garcia-Ojeda, J. C.; Deloach, S. A.; Robby; Oyenan, W. H.; Valenzuela, J. O-Mase: A Customizable Approach to Developing Multiagent Development Process. In: 8th International Workshop on Agent Oriented Software Engineering (AOSE) (2007)

[9] Guessoum, Z; Cossentino, M.; Pavón, J.: Roadmap of Agent-Oriented Software Engineering – The AgentLink Perspective. In: F. Bergenti, M. P. Gleizes, & F. Zambonelli (Eds.), Methodologies and software engineering for agent systems, Kluwer Academic Publishers, 431--450 (2004)

[10] Harmsen, A.F.: Situational Method Engineering. Moret Ernst & Young (1997)

[11] Hübner, J., Sichman, J.; Boissier, O.: A model for the structural, functional, and deontic specification of organizations in multiagent systems. In: Bittencourt, G. & Ramalho, G. L. (Eds.), 16th Brazilian Symposium on AI, SBIA'02, LNAI 2507, Berlin: Springer, 118--128 (2002)

[12] Karlsson, F.: Method Configuration - Method and Computerized Tool support. Doctoral Dissertation Dept. of Computer and Information Science, Linkoping University (2005)

[13] Lemaıtre, C.; Excelente, C. B.: Multi-agent organization approach. In: 2nd Iberoamerican Workshop on Distributed AI and MAS, Toledo, Espana (1998)

[14] OMG. Object Management Group. Software & Systems Process Engineering Meta-Model Specification, version 2.0. OMG document number: formal/2008-04-01 (2008) Available on http://www.omg.org/spec/SPEM/2.0/PDF.

[15] Pavon, J.; Gomez-Sanz, J.; ; Fuentes, R.: The Ingenias Methodology and Tools. In: Henderson-Sellers, B., Giorgini, P. (Eds.), Agent-Oriented Methodologies, Idea Group Publishing, 236--276 (2005)

[16] Rougemaille S.; Migeon, F.; Millan, T.; Gleizes, M.-P.: Methodology Fragments Definition in SPEM for Designing Adaptive Methodology: A First Step. In: Luck, M.; Gomez-Sanz, J.J. (Eds**.):** AOSE 2008**,** LNCS, vol. 5386, Springer, Heidelberg, 74–85 (2009)

[17] Stamper, R.: Signs, Norms, and Information System. In: Holmqvist, B.; Andersen, P. B.; Klein, H.; Posner, R. (Eds) Signs at Work: Semiosis & Information Processing in Organizations, Walter de Gruyter, Berlin, 349--397 (1996)

[18] Wood, M.; DeLoach, S. A.: An overview of the multiagent systems engineering methodology. In: 1st International Workshop on Agent-Oriented Software Engineering. LNCS, vol. 1957, 1--53 (2001)

[19] Zambonelli, F., Jenningns, N. R.; Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transaction on Software Engineering and Methodology, vol 12(3), 417--470 (2003)

Organizations Partitioning Optimization

Ammar Lahlouhi

Computer science department, University of Batna, 05000 Batna, Algeria
ammar.lahlouhi@gmail.com

Abstract. In this paper we describe an original approach to improve the organizational development of multi-agent systems. The approach extends the usual organization definition by adding the partitioning criteria and constraints. A such extension allows organization partitioning automation by a coherent optimization of an initial obvious partitioning using an adapted heuristic. The specification of criteria and constraints improves the quality of the developed organizations and the automation alleviates the development task. The heuristic's adaptation is made by adopting a multi-criteria method and the optimization is based on a coherent evolution. The first allows to benefit from the naturalism of multi-criteria methods and to add unlimited number of criteria while the second supports preventive and corrective constraints. The purpose of this paper is to clarify the organization's extension and the heuristic's adaptation.

Key words: Multi-agent systems, role-based development, graph partitioning, coherent evolution, multi-criteria optimization

1 Introduction

The development of Multiagent systems (MAS) is intricate. The agent-oriented software engineering (AOSE) (see [1] and AOSE workshops, for a survey) aims at simplifying such development by producing related techniques, tools and methods. The most promising of them are based on organizational metaphor. Hard work is accomplished in several organizational methodologies (such as Gaia [7] and TROPOS [5]). In spite of such efforts, MAS development remains however difficult. This paper targets improving MAS organizational development.

An organization is seen as a collection of roles that will be played by agents [2]. The common process of organizational development includes two main stages: designing an organization responding to customer's requirements and then developing the MAS that implements such an organization. Realizing an organization is made by partitioning it in partitions of roles and associating agents to the organization's partitions. To be valid, rational and reflects naturally the organizational characteristics, the organization partitioning (OP) must be coherent and optimized. To be more precise, it must optimize some criteria while maintaining some constraints. The usual organizational development considers OP as domain dependent, i.e., it leaves implicit the criteria and constraints. Hence, the

developer cannot conduct an accurate validation of OP. In addition, optimizing coherently criteria (which can be numerous, various and conflicting) is difficult to perform rigorously in an empirical way. The developer will then consider the first discovered issue which can be neither coherent nor optimal; forgetting, frequently, the OP's validation, rationality and naturalism. OP is then source of difficulties in MAS development. Our objective in this paper is the OP's improvement in two directions. Firstly, we improve MAS quality (validation, coherence ...) by making explicit the partitioning criteria and constraints. The usual organization definition is then extended by the definition of partitioning criteria and constraints. More details on a such extension are given in section 2. Secondly, we offer a support of automating OP and then freeing the developer from such partitioning.

In this paper, we consider a representation of an organization as a graph G=(V, E) where V is the set of vertices representing the organization's roles and E is the set of edges representing the relations between these roles. Such a representation allows us to consider OP as graph partitioning (GP). GP is an important problem with extensive applications to many areas. It is well known as NP-hard. Due to its importance, many efforts have been made to develop efficient heuristics and approximation algorithms. The GP problem is defined as follows: Given a graph G=(V, E), split V into k (k > 1) non empty pairwise disjoint sets (called partitions) V1, V2...Vn covering V. The goal of GP is to minimize the number of edges of E whose incident vertices belong to different partitions (referred to as edge-cut). For instance, in the application of GP to parallel computing, the nodes represent tasks and the edges represent communications between these tasks. The goal is to partition the tasks in k (k is the processors number) equilibrated (approximately equal size of tasks by processor) partitions while minimizing the edge-cut. The proposed solutions for such a problem can be classified into three classes [3]. Firstly, the algorithms to be executed off line either at compile or load time, secondly, those to be executed at run-time, and thirdly, those mapping the solution domain of partial differential equations to parallel computers. Heiss and Schmitz have proposed the particles' approach (PA) [4] which combines the benefits of all three classes. In addition, it avoids local minima. We use then PA to automate OP. However, some OP's particularities, such as the development context, prevent the direct use of PA. Consequently, we adapted PA to OP's context.

The remainder of the paper is organized as follows. Section 2 develops the organization definition's extension and the PA's adaptation. Section 3 concludes the paper.

2 Organization Partitioning

A graph modeling similar to that used in parallel computing can be used to model OP. The processors and their communications represent the agents and their communications. The tasks and their communications represent the roles and their relations. However, several dissimilarities prevent the direct use of

existing approaches of GP. Then, a GP approach (such as PA [4]) must be adapted before applying it to OP. The remainder of this section explains the OP's basics and the PA's adaptation. The PA's algorithm is given in Fig. 1 (see [4] for further details).

2.1 Constraints and Criteria

A GP can be considered as criteria optimization under constraints. For instance, PA [4] considers the parallel computer graph as an implicit (hard coded) predefined constraint, and load balancing... as criteria (which are combined to constitute one criterion, the resulted force). In the organization's partitioning (development context), the criteria can be varied, conflicting and of different scales. They cannot be then pondered to constitute one function as made in PA. They are domain knowledge and will be specified by the developer at the organization's development. Consequently, they cannot be hard coded (implicit) in the OP algorithm. The constraints will be described as logical expressions and the criteria as mathematical functions (to be optimized).

Code	**Comments**

cycle
wait for load change
if load increase then
M:= $\{i \in T / x_i = 1\}$
loadbalanced := false
while M $\neq \phi$ and not loadbalanced do
get information from neighbors	

for all i ∈ M do for all k ∈ neighbors do calculate f(i,k)	Functions evaluations
cand:= i $\underbrace{with}_{i \in M, k \in neighbors}$ max $\{f(i,k)\}$ direct:= k $\underbrace{with}_{k \in neighbors}$ max $\{f(cand,k)\}$	Choosing candidate node
if f(cand, direct) > 0 then migrate(cand, direct) send new load data to neighbors	Conditional migration
else loadbalanced := true else send new load data to neighbors end cycle	

Fig. 1. Outline of the load balancing algorithm of PA [4]. The comments indicate the parts that will be adapted in OP

2.2 Coherent Evolution

Optimizing a partitioning P is a P's evolution preserving the organization's coherence and optimizes the criteria. A partitioning evolution consists of a combination of coherent operations of creation, removing and / or modifying a partition p_i. The creation of p_i starts by creating an empty set p_i before moving a role from another partition p_j to p_i. Removing p_i will be made after moving the last role from p_i to another partition p_j. Finally, the modification of p_i will be made by moving a role from p_i (to p_j) or inversely (from p_j) to p_i. In addition to the creation and removing empty sets, it is clear that coherently moving roles between partitions is a fundamental process of OP's optimization.

Moving a role between two partitions can break some organization's constraints. Two strategies can be used to preserve the organization's coherence, preventive and corrective. The first strategy authorizes an evolution if it is coherent otherwise prevent it. The second strategy authorizes the evolution and then executes a corrective process to re-establish the coherence. The process of coherent moving roles must include the two strategies as follows. Verify the prevented incoherences. If a such moving is coherent (regarding the preventive constraints) then move the role and then re-establish the coherence (regarding the corrective constraints).

2.3 Coherent Optimized Evolution

The OP optimization uses an adapted PA [4]. In the following, we explain three fundamental adaptations we made to PA. Firstly, PA assumes an initial preexisting graph (architecture) between the partitions (processors) which is that of the parallel computer. In OP, such an initial graph does not exist. In the particular case of OP, we exploited the fact that each role is designed so it can be coherently assumed by an agent without requiring from that agent to assume other roles. A simple coherent obvious solution for the initial partitioning consists then of one role by partition.

Secondly, PA considers the optimization of one function (mono-criteria optimization); what is insufficient for OP which can require optimization of no fixed number of various conflicting criteria. To deal with a such situation, we use multi-criteria optimization [6]. In multi-criteria optimization, the simple order relation of mono-criteria optimization is replaced by that of dominance. Instead of seeking the partition with the maximum value, we seek the one with dominating value. A partition X dominates another partition Y if the two following conditions meet: (1) X is not worse than Y for all criteria ($\forall k \in \{1, 2 \dots m\}\ f_{gk}^X(t_i) \leq f_{gk}^Y(t_i)$), and (2) X is strictly better than Y for at least one criterion ($\exists k \in \{1, 2 \dots m\}\ f_{gk}^X(t_i) < f_{gk}^Y(t_i)$). The set of the partitions not satisfying one of the two previous conditions are incomparable; they are referenced as Pareto front.

In PA, the process of evaluating criteria consists of computing one function for each role and node. With the multi-criteria approach, we must evaluate several functions. In addition, PA chooses a destination node with the maximum value

of the function (force) but it must be positive since the negative one is directed to the source node itself ("no migration" is necessary). In the multi-criteria approach, we consider the source node s as any node but the values of its criteria are null since no change will be made in the criteria values without migration. We initialize then the functions of each role r for s by f(r, s) $=\langle 0, 0 \dots 0\rangle$. Finally, if there are several incomparable nodes, we choose one of them randomly.

Thirdly, PA assumes that moving roles between partitions is conditioned by the criteria optimization only since the constraints are hard coded in PA. However, OP requires the coherence verification for some constraints (before role moving), and the coherence re-establishing for others (after role moving). We add then to the migration process of PA a preprocess "constraints verification" and a post-process "coherence re-establishing" for moving a role. The preprocess starts by choosing the dominating node N and removes it from the list of considered nodes. If moving the role to N does not verify the constraints, the process chooses another node, and so on. In all cases, a node will be chosen (at least the source node s, which is considered as any node) since leaving the role in the source node is always coherent where the evolution process starts from a coherent partitioning. The equilibrium will be reached when the source node dominates.

The PA adaptation is outlined in Fig. 2. The adapted parts are indicated by comments in Fig. 1 and Fig. 2.

Code	**Comments**
Functions evaluations	
for all i ∈ M do	Multi-criteria
f(i, s)= $\langle 0, 0 \dots 0\rangle$	functions evaluations
for all k ∈ neighbors do	
f(i, k) = $\langle f_1(i,k), f_2(i,k) \dots f_n(i,k)\rangle$	
Choosing candidate node	
repeat	
cand:= p $\underbrace{with}_{i \in M \wedge i \neq p \wedge k \in neighbors}$ not dom (f(i,k), f(p,k))	Searching
direct:= q $\underbrace{with}_{k \in neighbors \wedge k \neq q}$ not dom (f(cand,k), f(cand,q))	non dominated node
chosen:= verify-constraints (cand, direct)	verifying the constraints
remove f(cand, direct)	
until chosen	
Conditional migration	
if direct ≠ s then	if chosen node is not
migrate(cand, direct)	the source s, migrate and
re-establish-coherence (cand, direct)	re-establish coherence

Fig. 2. Adaptation of PA algorithm of Fig. 1 to OP

3 Conclusion

The work presented in this paper can be considered as an improvement of GP methods. However, our goal is its embedding in MAS development process. In the development environment, our choice of PA has several advantages. For instance, it can be used with continually evolving organization since it endorses adding roles at run time.

The work presented in this paper includes several originalities. The organization definition's extension with partitioning criteria and constraints is made at the first time in this paper. The extension improves the MAS quality and allows automating OP freeing then the developer from OP. The automation is made by coherently optimizing an initial "simple obvious coherent" partitioning using an adapted PA. Such an automation is also made for the first time in this paper. The adaptation is based on making explicit the partitioning constraints and criteria, adopting multi-criteria optimization, and managing the evolution's coherence. Unlimited number of constraints and criteria (which can be conflicting) can be defined at the development time (not hard coded) and taken into consideration by the adapted PA. The latter endorses coherent evolution with preventive and corrective constraints.

It is worth noticing that further effort is needed to complete this work. For instance, it is important to define (or choose) a language (and an associated compiler or interpreter) and a methodology for expressing criteria and constraints. It is important to note that this work will be extended to support adaptive systems development. A such adaptation will be based on programming software processes.

References

1. Bergenti, F.; Gleizes, M.-P. and Zambonelli, F.: Methodologies and software engineering for agent systems: The Agent-Oriented Software Engineering Handbook. Kluver academic publisher (2004)
2. Ferber, J.; Stratulat, T., and Tranier, J.: Towards an integral approach of organizations in multi-agent systems: the MASQ approach. In: Multi-agent Systems: Semantics and Dynamics of Organizational Models, Virginia Dignum (eds.), IGI (2009)
3. Heiss, H.-U. and Dormanns, M.: Partitioning and Mapping of Parallel Programs by Self-Organization. Concurrency-Practice and Experience 8 (9) pp. 685-706 (1996)
4. Heiss, H.-U. and Schmitz, M.: Decentralized Dynamic Load Balancing: The Particles Approach. Information Sciences 84(1-2), pages 115-128 (1995)
5. Kolp, M.; Giorgini, P. and Mylopoulos, J.: A goal-based organizational perspective on multiagent architectures. In: Intelligent Agents VIII: Agent Theories, Architectures, and Languages, LNAI, vol. 2333, pp. 128-140, Springer, Heidelberg (2002)
6. Pomerol J.-Ch.: Multicriteria DSSs: State of the art and problems. Central European Journal for Operations Research and Economics, vol. 2(3), pp. 197-212 (1993)
7. Wooldridge, M.; Jennings, N. R. and Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (3) pp. 285-312 (2000)

Engaging Stakeholders with Agent-Oriented Requirements Modelling

Tim Miller[1], Sonja Pedell[2], Leon Sterling[1], and Bin Lu[1]

[1] Department of Computer Science and Software Engineering
[2] Department of Information Systems,
University of Melbourne, Parkville, 3010, VIC, Australia

Abstract. One advantage of using the agent paradigm for software engineering is that the concepts used for high-level modelling, such as roles, goals, organisations, and interactions, are accessible to many different stakeholders. Existing research demonstrates that including the stakeholders in the modelling of systems for as long as possible improves the quality of the development and final system because inconsistencies and incorrect behaviour are more likely to be detected early in the development process. In this paper, we propose three changes to the typical requirements engineering process found in AOSE methodologies, with the aim of including stakeholders over the requirements engineering process, effectively using stakeholders as modellers. These changes are: withholding design commitment, delaying the definition of the system boundary, and delaying the stakeholder "sign-off" of the requirements specification. We discuss our application of these changes to a project with an industry partner, and present anecdotal evidence to suggest that these changes can be effective in maintaining stakeholder involvement.

1 Introduction

In software engineering, product and project stakeholders are a valuable resource for eliciting and validating requirements. Stakeholders are especially important for socio-technical systems, in which the interaction between people and technical systems can form behaviour outside of the control of the technology itself.

The agent paradigm recognises that most stakeholders are non-technical, so by using concepts such as roles and goals, which are palatable for most people, stakeholders can provide feedback on models early in the development process. As a result, artifacts in agent-oriented software engineering play a somewhat different role to other types of artifacts. As well as documenting the requirements engineers' understanding of the domain, which requirements specifications typically do, they can also be used to encourage rich discussion between stakeholders, including requirements engineers.

Many requirements engineering processes, including those in agent-oriented software engineering methodologies, aim to define the interface and product features, and to precisely specify and validate these as early as possible in the development lifecycle. Our view is that, while making these decisions early has

benefits, premature commitment to certain solutions and definitions may discourage stakeholders that do not agree with or understand these decisions from participating in conversations with system developers. We advocate involving stakeholders in the development process for as long as possible, to continue engaging them in rich conversations that can help understand and define the system.

For engineering socio-technical systems, we propose small changes in the typical requirements engineering process found in software engineering (including AOSE) methodologies, with the aim of promoting conversation between stakeholders. The changes are based on results from existing research, which is discussed in Section 3. The proposed changes are:

1. Withholding design commitment by allowing inconsistencies and ambiguities early in the requirements engineering process. This allows different viewpoints of stakeholders to be represented, encouraging them into conversations for longer than they otherwise may. We are not the first authors to take this stance. For example, Easterbrook and Nuseibeh [5] discuss a framework with the purpose of allowing and dealing with different stakeholders' viewpoints. Paay et al. [15] suggest that withholding design commitment encouraged conversations between different stakeholders.
2. Delaying the definition of the system boundary. By defining the system boundary early in the process, some solutions may be eliminated before they can be discussed by the stakeholders, even though they may be more suitable than the remaining solutions.
3. Delaying the "sign-off" of requirements (or the end of the requirements engineering process) until the high-level agent design. That is, the requirements are considered only complete once we identify which agents are to be built and what their behaviour is to be. This is related to the second point, as it also helps to define the system boundary.

It is our view that these changes can be used in any agent-oriented development methodology, and are useful for breaking down barriers between stakeholders and software engineers, especially for social-technical systems. In Section 4, we present the application of these changes to an industry case study, and discuss the advantages and disadvantages that resulted from these changes. The goals of the paper are to present these processes to researchers and practitioners in agent-oriented software engineering in order to promote discussion and receive feedback on these ideas.

2 Agent-Oriented Requirements Engineering

With the agent paradigm increasingly becoming a popular and successful way for modelling complex systems [14], methodologies for agent-oriented software engineering have become an important research field. Several such methodologies have been proposed, such as Tropos [2], Prometheus [16], Gaia [22], INGENIAS [17], and ROADMAP [10].

The typical requirements engineering process in these methodologies involves the following steps[3]:

1. elicit requirements from the stakeholders on the project;
2. derive scenarios that specify typical usage of the system;
3. define the system boundary;
4. define the environment;
5. derive a goal model outlining the major goals of the system;
6. define the role descriptors for the roles that will help to achieve the system goals;
7. define the interaction model, which specifies how roles in the system will interact; and
8. iterate over steps 1-7 with stakeholders until a shared understanding of the system is reached.

Although agent methodologies do not discuss requirements sign off, they define the *software requirements specification* (SRS) as the combination of the system boundary, goal models, role, and interaction models. From this, we infer that the major stakeholders would sign off on these documents after step 8. This would form the basis of a contract for the system development to proceed.

Variations of these steps are possible; for example, the Gaia methodology defines preliminary version of the role and interaction models as requirements, and more detailed definitions as architectural design; and Prometheus defines interaction models as architectural design.

From this point in the development process, agent-oriented methodologies typically treat subsequent tasks as design-level, so stakeholder input would not be required. The tasks include defining the agent types in the system, which agent types will play which roles, the activities that the agents will perform (these activities will both fulfill the agent's role and the goals related to that role), and implementing and testing the agents.

2.1 Modelling with Roles and Goals

The work in this paper builds mainly on the work of Sterling and Taveter [19]. Their work has focused on how to make high-level agent-oriented models palatable to non-technical stakeholders. This is achieved using role and goal models with a straightforward and minimal syntax and semantics.

Goal models are useful at early stages of requirements analysis to arrive at a shared understanding [11, 9]; and the agent metaphor is useful as it is able to represent human behaviour. Agents can take on roles associated with goals. These goals include quality attributes that are represented in a high-level pictorial view used to inform and gather input from stakeholders. For example, a role may contribute to achieving the goal "Release pressure", with the quality goal "Safely". We include such quality goals as part of the design discussion

[3] Some methodologies do not strictly follow this process, but this is a good approximation of all methodologies.

and maintain them as high-level concepts while eliciting the requirements for a system. For this purpose the AOSE goal models have to be simple yet meaningful enough to represent the goals of social interactions.

Figure 1 shows the syntax employed by Sterling and Taveter, which we have used in our work. Goals are represented as parallelograms, quality goals are clouds, and roles are stick figures. These constructs can be connected using arcs, which indicate relationships between them. Figure 1 shows a high-level role and goal model from our industry project of an aircraft turnaround simulator. This system simulates the process of multiple aircraft landing at a single airport, allowing one to experiment with resource allocation. The goal *Aircraft Turnaround* is the highest-level goal, and the sub-goals below this contribute to fulfill the higher-level goal. The quality goal *Efficient* specifies that goal *Aircraft Turnaround* must be satisfied with the quality attribute *Efficient*. The roles plays some part in bringing about the goal *Aircraft Turnaround*.

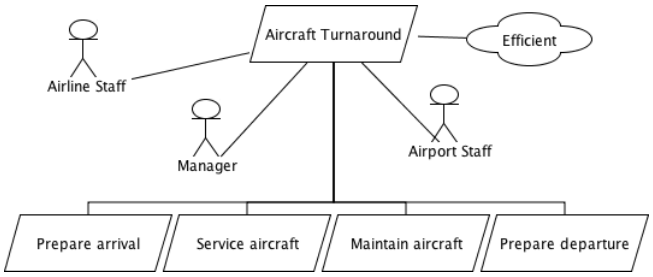


Fig. 1. An excerpt for the high-level goal on the aircraft turnaround project.

It is important here to note that the semantics described above is a complete definition of Sterling and Taveter's goal models, leaving space for interpretation of the model. This helps to engage stakeholders who have no experience in agent modelling, and encourages round-table discussion between stakeholders and requirements engineers.

3 Changing the Agent-oriented Requirement Engineering Process

The changes presented in this paper are based on existing research in software engineering and interaction design, however, it is our view that the agent paradigm offers certain unique capabilities to the requirements engineering process that other paradigms to not. In this section, we motivate and justify our reasons for modifying the requirements engineering process, link this to existing literature that provides evidence to confirm our hypothesis, and discuss why the agent paradigm is particularly suited to these changes.

At first sight, delaying clear definitions seems antithetical or uncommon to the routines of software engineering, which is typically a structured process aimed at removing ambiguity and deriving clear definitions as early as possible in the development process. However, a body of literature that looks at software engineering from a social science perspective recognises that models and other documentation in software engineering have been used as a way to think through problems, to reach agreements, and to elaborate the needs of stakeholders in a different way than simply feeding into a formal process of modelling for system design [13, 3, 18]. For example, a goal and role model serves a different purpose for a designer than for a domain expert.

3.1 Withholding Design Commitment

The first change to the requirements engineering process is to withhold the commitment of system designs. By this, we mean holding off any particularly functional details of the system that fulfill the user requirements. At the early stages of requirements elicitation, we may not be able to clarify social concepts sufficiently to resolve uncertainty. For example, in a business domain, roles such as manager, researcher, and team leader can be well defined. However, in a social domain, roles may not be so straightforward to define. Consider trying to define the role of a grandparent, and the goals that role may want to achieve. As a result, we advocate that the social goals related to these concepts should be modelled ambiguously, even to the point where formal documents are written.

Quality requirements at the early stages of elicitation tend to be imprecise, subjective, idealistic and context-specific, as discussed by Jureta and Faulkner [11]. Garcia and Medinilla [6] describe high-level quality goals as a specific form of uncertainty that can be used as a descriptive complexity reduction mechanism and to model and discuss uncertainties in the environment. In our requirements elicitation process, we seek complexity reduction without losing the richness of the concepts themselves. Instead of eliminating uncertainty early in the process, we embrace it and withhold design commitment, at least until there is clarity and understanding between stakeholders of what it may mean to disambiguate [7].

High-level goals associated with activities can act as a point of reference for discussing the usefulness of design alternatives to achieve these goals instead of a decomposition into single requirements. The multi-agent paradigm offers benefits over other paradigms because the concepts used in modelling, such as roles, goals, and interactions, are part of every day language. Real organisations consist of roles, and specific people fill these roles each day, including stakeholders in a software engineering project. As such, stakeholders are familiar with these concepts, and are comfortable talking about them.

3.2 Delaying the Definition of the System Boundary

In many software engineering processes, the system boundary is defined before requirements analysis takes place. Often, this is one of the first agreements made between clients and software engineers.

Gause and Weinberg [8] found that natural subconscious disambiguation is one of the most common sources of requirements failure. In this situation, unrecognized or unconsciously assumed, incorrect meaning finds its way into the specification [1]. The problem is compounded by the fact that not only do software engineers consciously try to resolve uncertainty early in the process, before its impact on design is completely understood, they may also do this subconsciously. More importantly, checking the absence of requirements once we have a formal specification document is likely to be more difficult, because these documents are typically highly technical, and there less accessible to the stakeholders [12].

Once the boundaries of a system are defined, the focus of attention is within these boundaries; solutions beyond this boundary are no longer considered. Such a restriction discounts solutions that may be more suitable, and is more likely to result in some stakeholders losing interest in the project if their desired solution falls outside of these boundaries.

This does not imply that one should not be thinking about the system boundary. Specifically, all stakeholders should be aware of any other systems that may be used as part of the solution to the domain problem.

The multi-agent paradigm is well suited for such models, because high-level role and goal models can be discussed and modified without defining the system boundary, while still allowing all stakeholders to come to a shared agreement of what the entire socio-technical system will comprise.

3.3 Delaying the "Sign-off" of Requirements

The sign-off of the SRS often forms part of a contractual agreement between clients and developers. The SRS defines the external interfaces to a software system and provides a complete description of the extended behaviour of the software.

In the process of software engineering, the sign-off of a requirements specification is generally performed before any high-level design takes place. If left until after design commences, developers may unnecessarily waste time on design tasks, only to find the requirements have changed.

In the multi-agent domain, we advocate delaying the sign-off of the SRS by stakeholders until as late as possible before it impacts architectural design. This allow discussions to continue between stakeholders for a longer period. Furthermore, it also helps stakeholders to understand the proposed behaviour of the system, because role and goal models define motivation, not behaviour.

3.4 Discussion

The first two changes proposed in this section are not new in the social domain. Our work is consistent with results from researchers cited in the previous sections. As far as the authors are aware, the third change, delaying the sign-off, has not been investigated before.

While we present these three proposed changes as being separate changes, they are in fact, closely related. By not defining the system boundary, we are in fact withholding design commitment. Similarly, by not signing off on the SRS early, we are leaving open design decisions, thereby withholding design commitment.

These changes are presented separately because we view them as different tasks. Withholding design commitment is a general approach in which we do not take design decisions too early, but in general, the requirements elicitation process will run in the same order. However, the definition of the system boundary is a specific task that we aim to put later in the requirements engineering process. Typically, defining the system boundary is one of the first tasks performed in requirements engineering, and this is suitable for most business applications. However, for socio-technical systems, we see that a benefit in delaying the definition of the system boundary until after we fully understand the behaviour of the *entire* socio-technical system, including humans and external systems, not just the software system being built.

4 Experience

In this section, we present our experience on a project involving an industry partner. We discuss how the changes were achieved in an industry project, what effect they had on the project, and how other stakeholders responded to them.

4.1 The Project

The project is a joint project between the University of Melbourne and Jeppesen, a company that specialises in aeronautical services. The goal of the project is to construct simulation software for air traffic management using the agent paradigm as the modelling tool. The particular project on which we applied the modified requirements engineering process was a simulation of aircraft turnaround. This system simulates the process of multiple aircraft landing at a single airport, and how resources (including staff) could be allocated to efficiently turn around the aircraft, including re-stocking supplies, as well as cleaning, repairing, and maintaining the aircraft.

The major stakeholders of the project were our research team and a group of software engineers at Jeppesen who had no significant exposure to agent-oriented modelling in the past.

Figure 1 (in Section 2) shows part of the high-level role-goal model for the aircraft turnaround project. In this figure, the high-level goal of turning around

the aircraft is achieved by the four subgoals of preparing for arrival, servicing the aircraft, maintaining the aircraft, and preparing for departure. The roles of *Airline Staff* and *Airport Staff* in this figure are in fact *aggregate roles*; that is, they are sets of roles, such as aircraft maintenance engineers, cleaners, and airline crew, which are described in lower-level role-goal models. The *Manager* role is responsible for overseeing the entire turnaround and re-allocating resources if there is a delay in turning around one aircraft.

4.2 Withholding Design Commitment

The requirements elicitation proceeded by our group being given an overview of the aircraft turnaround process, including the staff involved, and constructing a high-level goal and role model that represented our understanding of the system. These diagrams were improved and refined over a series of six round-table meetings with the stakeholders, in which the role and goal models were distributed to each stakeholder before a meeting, and were then used as shared artifacts to guide conversations. Over the course of these meetings, other models including the interaction models, environment models, and agent types were progressively introduced as we gained further understanding of the system.

Withholding design commitment was achieved by basing conversations between stakeholders on the role-goal models and using the role-goal models as a facilitator to open up the discussion. In this regard, the goal models took a similar role as the guiding rules described by Tjong et al. [20], whose aim is to detect uncertainties in order to trigger questions to be asked of the client.

The role and goal models were helpful in triggering communication about the specific challenges of the domain, and for identifying missing parts of the system. For example, one stakeholder commented from a single glance at the high-level goal model that air traffic controllers play a role in aircraft turnaround, and this induced discussion about how the system should handle new traffic entering the airport. In subsequent iterations, the air traffic controller role was deemed unnecessary for the system and was dropped, but changes related to this remained.

Our experience indicates that having models evolve over time lead to a clearer solution, as early concerns regarding concepts such as resources were delayed without jumping to a pre-conceived solutions. Later in the development process, successive versions of the models were used as a reminder to the design decisions that were made. This gave the research team something to fall back on when discussions started to get too complex for some stakeholders or drifting off from original high-level goals. The example of the air traffic controller role illustrates this, in which the models were updated to reflect this role, but even after its removal, parts of the model related to it remained. This is consistent with the findings described by MacLean and Bellotti [13].

Our industry partners are comfortable with the role and goal models, although this is perhaps to be expected as they are software engineers. However, Paay et al. [15] have used role and goal models as shared artifacts in the social-

technical domain with non-technical stakeholders such as ethnographers to similar effect.

4.3 Including Agent Types as an SRS

We delay the system boundary definition and the SRS sign-off using the same technique: by leaving both until the high-level design.

The major divergence we take from the typical AOSE methodology is to include the agent types, including the activities they perform, as part of the SRS. As discussed in Section 2, methodologies typically use roles, goal, and interaction models as requirements, while agent types are part of the architectural design.

In this project, the SRS consisted of the role and goal models, the interaction models, the environment model, and the agent types. Combining the environment model and the agent types defines the functionality of the system, while the role and goal models, and the interaction models, help to motivate this functionality. For this particular simulation system, there was a one-to-one mapping between roles and agents.

Signing-off on the SRS We believe that roles, goals, and interactions do not provide sufficient detail to define system behaviour. While role and goal models specify the goals that the system will achieve, and the roles (and their responsibilities) that will help to achieve them, they do not define functionality; that is, how the system will behave to achieve these goals. For example, the model in Figure 1 specifies the goals that need to be achieved to turnaround the aircraft. Role descriptors for the three roles in this figure outline the responsibilities to ensure the turnaround goals are achieved. However, this does not define which activities will be performed to achieve the goals. In some cases, one can extrapolate the activities from the responsibilities and goals, but this is not always the case.

Our approach of including the environment model and agent types, including activity descriptions and their effect on the environment, specifies the behaviour of the system. As such, these activity descriptions are similar to functional requirements that one would find in a non-agent-based SRS, and it is at this point that the major stakeholders will be able to sign-off on the models.

A sign off is an agreement that overall goals are important, and that the defined system will achieve these goals. We came to a solution that all were satisfied with. We see this as a benefit in itself.

Furthermore, the stakeholders commented that the behaviour of the system was clearer when the agent types were included, even though the mapping from roles to agents was one-to-one. This is perhaps partly due to the similarity between activities and functional requirements, but the stakeholders commented that this was due to the fact that they were able to make a clear judgement as to whether the behaviour fulfilled their expectations. In our view, this justifies the decision to include the agent types in the SRS.

Defining the System Boundary Including agent types in an SRS has a second effect: it completely defines the system boundary. Role and goal models define the entire socio-technical system, with no commitment to which roles will be played by which agents. As Cheng and Atlee [4] discuss, integrated systems pose problems in defining the system boundary, which can be solved by assigning responsibilities to different parts of the system, including the software system being constructed, human operators/users, and external systems. Our notion of a system boundary is exactly this: by describing the responsibilities of roles in the entire system, we can define the system boundary by specifying which agents will fulfill which roles, whether these agents are software agents, humans, or external systems.

For example, in Figure 1 we can define one system boundary by specifying that software agents will play all of the relevant roles, making the system a complete simulator of the turnaround process. Alternatively, we can define another system boundary in which the *Manager* role is played by a human. One can see that assigning one role to a human instead of an agent changes the system and its interface greatly. In the first instance, the system is a complete simulation of the aircraft turnaround process. In the second instance, the result is an interactive system in which managers are able to assess different resource allocation mechanisms.

In this project, the system boundary was left undefined for most of the requirements elicitation process. The stakeholders were comfortable with the lack of a system boundary, and this was not explicitly mentioned to them during the requirements elicitation. However, as software engineers themselves, they did not see any great benefit for this project, because they felt only one system boundary was sensisble. However, they also did not find that it was detrimental to the project. We did not find that delaying the definition of the system boundary had any adverse effects on the progression of the system, although this was not a controlled experiment. In addition, we found that conversations about the system, including details about roles and goals, continued after the agent types had been assigned, due to the system functionality becoming clearer.

To our group, the benefits of not defining a system boundary are illustrated by the project. The system was intended to be a simulation of the air traffic turnaround domain, with all roles, including those in Figure 1, being played by software agents (the first boundary in the previous paragraph). One discussion that took place late in the requirements elicitation process indicated that there may have been scope for the system boundary to be changed to the second boundary, in which the *Manager* role is partly played by a human. Had the system boundary been defined at the start of the requirements elicitation, this discussion may not have taken place.

5 Conclusions and Related Work

AOSE models are useful as a shared artifact for communication between stakeholders and software engineers. We find that using the agent-oriented models of

Sterling and Taveter [19] as part of requirements elicitation allows meaningful conversations between all stakeholders about abstract concepts, with goals as the catalyst. The role of the goal models is not simply to lead to the development of a system, but also as a way to think through problems and to reach agreements. By making these accessible to all stakeholders, and by keeping stakeholders involved in discussions as long as possible in the requirements elicitation process, we aim to increase the quality of requirements specifications.

In this paper, we proposed three changes to the typically AOSE requirements engineering process that we believe help to engage stakeholders: 1) withholding design commitment; 2) delaying the definition of the system boundary; and 3) delaying the sign-off of the SRS to be as late as possible without affecting system development.

Our experience with an industry partner suggests that not committing to a specific design solution early in the requirements elicitation gave the team an opportunity to further explore and understand the specific challenges related to the high-level goals of the socio-technical system.

We propose delaying the definition of the system boundary and the signing-off of the requirements by including the agent types as part of the SRS. As far as we know, we are the first authors to consider this, rather than including these as part of the architectural or detailed design. By defining which agents will play which roles, we define the behaviour of the system, and implicitly define the system boundary. The experience with our industry partner indicates that this decision is justified.

Guizzardi and Pereni [9] have also recognised the importance of stakeholder involvement. Like us, they consider the goals of all stakeholders, and the interdependencies between these goals, as an initial step in understanding requirements. Yu [21] advocates the agent-oriented paradigm as a tool for helping to establish the *why* of a system, which helps stakeholders to understand the problem at hand. Similar to us, Yu uses high-level motivation models, in this case, specified in i^*, to share between stakeholders. The i^* models contain significantly more information than our motivation models, including concepts such as activities, resources, and dependencies between all of these. We explicitly aim to reduce the number of concepts and the amount of syntax to keep models simple. Yu offers no specific *techniques* for engaging stakeholders, as the focus of the work appears to be on the tools and notations for doing so.

References

1. D. Berry, E. Kamsties, and M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity - a handbook version 1.0, 2000.
2. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
3. G. Button and W. Sharrock. Occasioned practices in the work of software engineers. In M. Jirotka and J. Goguen, editors, *Requirements Engineering: Social and Technical Issues*, pages 217–240. Academic Press, 1994.

4. B. Cheng and J. M. Atlee. Research directions in requirements engineering. In L. Briand and A. Wolf, editors, *Proceedings of the International Conference on Software Engineering*, pages 285–303, 2007.
5. S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, 11(1):31–43, 1995.
6. A. Garcia and N. Medinilla. The ambiguity criterion in software design. In *International Workshop on Living with Uncertainties*. ACM, 2007.
7. D. Gause. User driven design – the luxury that has become a necessity, a workshop in full life-cycle requirements management. In *ICRE 2000, Tutorial T7*, 2000.
8. D. Gause and G. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House Publishing Co., Inc., New York, NY, USA, 1989.
9. R. Guizzardi and A. Perini. Analyzing requirements of knowledge management systems with the support of agent organizations. *Journal of the Brazilian Computer Society (JBCS)-Special Issue on Agents Organizations*, 11(1):51–62, 2005.
10. T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the Gaia methodology for complex open systems. In *Proceedings of the First Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 3–10. ACM Press, 2002.
11. I. Jureta and S. Faulkner. Clarifying goal models. In J. Grundy, S. Hartmann, A. Laender, L. Maciaszek, and J. Roddick, editors, *ER (Tutorials, Posters, Panels & Industrial Contributions)*, volume 83 of *CRPIT*, pages 139–144, 2007.
12. E. Kamsties, D. Berry, and B. Paech. Detecting ambiguities in requirements documents using inspections. In *Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01)*, pages 68–80, 2001.
13. A. MacLean, V. Bellotti, and R. M. Young. What rationale is there in design? In D. Diaper, D. J. Gilmore, G. Cockton, and B. Shackel, editors, *Proceedings of the 3rd Int. Conf. on Human-Computer Interaction*, pages 207–212, 1990.
14. S. Munroe, T. Miller, R. Belecheanu, M. Pechoucek, P. McBurney, and M. Luck. Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *Knowledge Engineering Review*, 21(4):345–392, December 2006.
15. J. Paay, L. Sterling, F. Vetere, S. Howard, and A. Boettcher. Engineering the social: The role of shared artifacts. *International Journal of Human-Computer Studies*, 67(5):437–454, 2009.
16. L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A practical guide*. John Wiley and Sons, August 2004.
17. J. Pavón and J. Gómez-Sanz. Agent oriented software engineering with INGENIAS. In *Multi-Agent Systems and Applications III*, volume 2691 of *LNCS*, pages 394–403. Springer, 2003.
18. D. Randall, J. Hughes, and D. Shapir. Steps toward a partnership: ethnography and system design. In M. Jirotka and J. Goguen, editors, *Requirements Engineering: Social and Technical Issues*, pages 241–254. Academic Press, 1994.
19. L. Sterling and K. Taveter. *The Art of Agent-Oriented Modelling*. MIT Press, 2009.
20. S. F. Tjong, M. Hartley, and D. Berry. Extended disambiguation rules for requirements specifications. In C. Alves, V. Werneck, and L. Marcio Cysneiros, editors, *Proceedings of Workshop in Requirements Engineering*, pages 97–106, 2007.
21. E. Yu. Agent-oriented modelling: software versus the world. In *Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 206–225. Springer, 2002.
22. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003.

Towards Requirement Analysis Pattern for Learning Agents

Shiva Vafadar, Ahmad Abdollahzadeh Barfourosh

Intelligent Systems Lab
Computer Engineering and IT Faculty
Amirkabir University of Technology
vafadar@aut.ac.ir, ahmad@ce.aut.ac.ir

Abstract. Learning is a capability that can be incorporated into software agents to handle the complexity of dynamic and unexpected situations, exploiting available artificial intelligence (AI) techniques. Despite design techniques for learning agents have been discussed in agent oriented software engineering literature, how to identify and analyze the requirements for learning agents is still poorly addressed. In this paper, we introduce a pattern for requirement analysis of learning agents. This analysis pattern contains a group of related, generic meta-classes of learning and their relations in a domain neutral manner which can be described as elements of conceptual modeling of learning requirement of agents. The applicability of the pattern has been investigated through the development of a book trading case study.

Keywords: Agent Oriented Software Engineering (AOSE), Analysis Patterns, Requirements Analysis, Learning

1 INTRODUCTION

Today software systems are used in more complex application domains such as web-base computational markets and distributed network management [1], which demand for software systems with autonomic properties [2]. This complexity often arises from open networked and heterogeneous environments with dynamic and unpredictable scenarios in which software is expected to operate. Enriching software systems with the capability of improving while operating can benefit from available artificial intelligence (AI), and agent oriented software engineering aims at providing methods to support developing systems with this property [1]. One of the capabilities which can help intelligent software agents to perform more appropriately in dynamic and volatile situations is learning. Issues in designing software agents with learning capabilities have been discussed [3,4] but techniques for requirement analysis of an agent's learning is still poorly addressed.

Requirements analysis addresses the identification and specification of the functional and non-functional (or quality) characteristics expected for the system to be developed, and analyze them in terms of ways to operationalize them. Therefore, requirements analysis activities encompass the problem domain as well as the solution domain, with the aim to provide effective information for the system design. More specifically, first activities of requirements analysis are focused on the problem domain analysis and the elicitation of expected features of

the system-to-be, but late requirements analysis activities focus on a deeper understanding of these system features, thus providing information for architectural and design concerns in terms of available candidate. The latter tries to provide required information for moving smoothly from requirements to high level design of the system. Taking the perspective of a requirements engineer who may not be expert in AI techniques, we believe that providing methods for supporting analysis of learning as one of the capabilities which help software agents to achieve their goals would be beneficial.

Based on this view, in this paper we present a pattern based approach for analyzing agent's learning capability. A main novelty of this work is its focus on the late requirements analysis phase of the development of learning agents, which abstracts from the learning approach that will be adopted in design and implementation. Analysis patterns are a group of related, meta-classes and their relations which present issues of conceptual modeling for analyzing requirements [5,6]. These objects are defined in a domain-neutral manner and they resemble the notion of chunks of formalized knowledge that are at a higher level of abstraction than individual classes. Our pattern can be considered as a guideline for modeling learning during requirement analysis of agent based systems. Using this pattern during late analysis activities can provide information to move easily from requirements to architectural design of software agents.

In order to evaluate the pattern we use a case study (which is a book trading system) and apply the pattern on it. By comparing the results of applying the pattern on the case study with required information for design and implementation of learner agent during software development, we can assess the applicability of our pattern on developed application.

In the following sections of this short paper, we present analysis pattern for learning agents and describe participants of the pattern. We also introduce our case study for investigating applicability of the pattern and discuss preliminary results.

2 Analysis Pattern for learning agents

In this section, we present our analysis pattern for learning agents. We suppose that an agent-oriented approach is used during requirements engineering process. In addition, it is supposed that during early analysis, requirements engineer has identified learning as a feature which assists agent(s) to achieve the goals. Having these prerequisites, for late analysis of the requirements, software engineer can use this pattern for modeling learning requirement of the agents. This pattern can be used as a guideline for modeling the learning as an AI capability which incorporated to agents. The result will be a model which helps stakeholders to understand learning-related issues of the system and making decision among multiple options which are available for designing and testing the learning of the agent.

Name: Learning Pattern

Classification: Analysis

Problem: How should an agent be analyzed in order to specify its learning capabilities?

Context: An agent-based system where a role needs to improve its performance while executing its tasks and getting experience. This role needs learning capability in order to carry out one or more tasks or achieving a goal.

> ***Domain of Application:*** the pattern is of a general nature. The domain of application is not specified.

Forces:

- Performing the task(s) or achieving the goal(s) is not possible without learning or learning makes it possible in higher quality or less time.
- Agent's knowledge (which can be related to the process of doing the task or primitive knowledge or rules) is not complete and it can be improved by getting experience.
- The agent can perform some tasks in order to get some experience or there is adequate training data which help the agent to improve its behavior.
- The agent can receive feedback from the environment after carrying out the task.

Solution: We suggest using the following model to analysis learning requirement. Figure 1 shows our solution to solve mentioned problem.

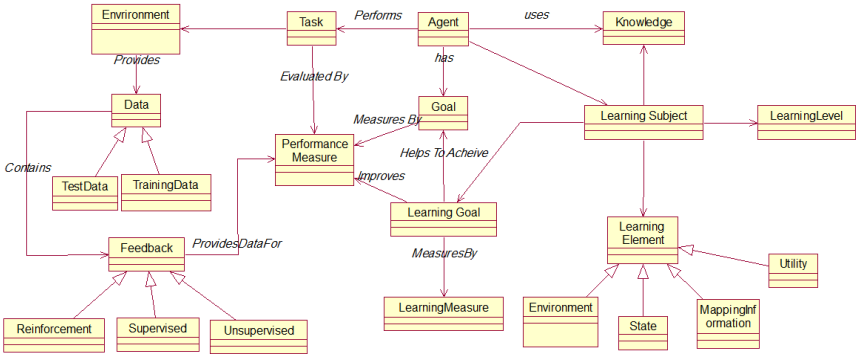


Fig. 1. Analysis Pattern for learning agent

As figure 3 shows, there are 13 participants in suggested model. In the following, we explain these meta-classes.

- **Agent:** Indicates the agent which its learning requirement is analyzed.
- **Goal:** Indicates the goal(s) that the agent is responsible for. Identifying the goal(s) of the agent is the first step of analyzing activity. Since agents are goal-oriented entities, the goals an agent tries to achieve plays vital role in agent analysis. Learning is a technique that can help the agent to achieve its goals. Therefore, the kind of the learning is expected from the agent is influenced by the goal(s) it is responsible for.
- **Task:** Specifies the tasks that the agent can perform. The relationship between tasks and learning is bidirectional. In one hand, we can specify the tasks the agent can perform and then identify which of them needs improvement by incorporating learning. At the other hand, we may know that to achieve its goals or improve its behavior agent must have learning capability. Therefore, we should recognize that what kind of tasks the agent should be able to perform to learn (such as the tasks required for exploration and experience generation). Both approaches are necessary for determining the required tasks.
- **Performance Measure:** embodies the criterion for success of an agent's behavior for achieving its goal. An agent has learning capability if its performance improves during performing the tasks. This improvement is measured by performance measure. Therefore, performance measure is a factor for evaluating learning capability of agent and it has an important role for defining learning goal.
- **Learning Goal:** Denotes improvement in agent's performance measure which is expected by incorporating learning. It also specifies in which duration this improvement

is expected. It affects on learning elements of the agent because the amount of improvement defines which parts of the agent should improve their behavior to attain learning goal. It is affected by many meta-classes such as agent goal, its performance measure, input data and its quality, feedback is available for the agent, tasks the agent can perform, prior knowledge and its quality. For example if the input data is not adequate or its quality is low, requirements engineer may decide extend duration which agent can achieve its learning goal.

- **Learning Subject:** Specifies the subject of the learning. It can be a concept of the environment or an internal state that agent capture knowledge about it.
- **Learning Element:** Defines issues that agent should learn to achieve learning goal. On the other hand, it defines learning goal in more details with respect to the learning subject such as: **State** which is mapping from conditions on the current state to the actions, **Environment** which is relevant properties of the world from percept sequence, **Mapping information** which is information about the way world evolves, results of possible actions the agent can take on the environment. **Utility** which is information indicating the desirability of the world state and action
- **Feedback:** Defines the type of the feedback is received by the agent which can be supervised, unsupervised or reinforcement. The feedback is one of the major issues that affect on selecting appropriate leaning algorithm during design. Therefore, during analysis we should specify what type of feedback is obtained for agent in the domain.
- **Knowledge:** Defines the agent's knowledge. It contains the knowledge the agent has prior to start his actions. This knowledge is defined according to the tasks the agent should perform, the knowledge that the agent expected to achieve during performing the tasks and the knowledge is required to achieve learning goal.
- **Environment:** Defines the environment the agent is acting on and all of its participants. Environment is an important factor in analyzing agent. How well an agent can behave depends on the nature of the environment. The properties of the environment such as fully observable vs. partially observable should be defined during analysis. These characteristics also influence learning algorithms which are selected during design. On the other hand, environment provides all the data that agent learns from. Therefore, the constraints of the environment vastly affect the constraints of the learning of the agent.
- **Data:** Defines the raw data that is received from environment (and all its participants) and is used as learning input. Therefore, it has a vital role in learning process. Amount of data and its quality has an important role for deciding about learning algorithm and it is an important criterion which affects our expectation from learning. Information which is related to data helps requirements engineer to decide about tradeoffs between duration and quality of learning. Test Data and Training Data are different types of data that should be considered during analysis.
- **Learning Measure:** Defines the measure for evaluating learning capability of the agent. It can be described by criterion such as preciseness and speed.
- **Learning Level**: Describes level of the learning expected from the agent. It can have a wide range from remembering the information to knowledge based inductive learning.

Resulting Context: The meta-classes of the model are domain independent conceptual classes that present abstract issues of learning capability. For analyzing various systems in different domains, it is just required to find the instances of each meta-class in the domain. These instances (or type of entities) and interaction between them construct the conceptual model of learning capability of the agent. By using analysis pattern for learning agent, analysis activity of learning capability can become easier because the pattern contains the issues that should be considered during analysis of learning.

Related Patterns: learning design patterns [3, 4]

3 Case Study: Book Trading System

In this section, we define our case study for evaluating applicability of the pattern. We also present the results of developing the case study and applying the pattern on it.

The case study that we select is a Book Trading System (BTS). Our system is an extension on Book Trading examples that comes with JADE 3.1. We modify the scenario as it includes some agents that sell books and other agents, which buy them on behalf of their users. Buyer's goal is purchasing the cheapest book while seller's goal is to achieve the highest profit. In this case, we also consider learning as an expected capability for the seller agent. Seller should explore various prices for each book and try to find the best price, which increases its profit.

Using an analysis pattern for modeling a system contains three main steps; retrieval, adaptation, and integration. In our case study, in retrieval step, we choose learning pattern for seller agent because learning is an expected capability of this agent. For adaptation, we instantiate conceptual classes of the pattern by recognizing related concepts in the application domain. The result will be the conceptual model of the seller agent, which contains learning related issue of the application. Figure 2 shows seller agent conceptual model, which is the result of applying the pattern on BTS.

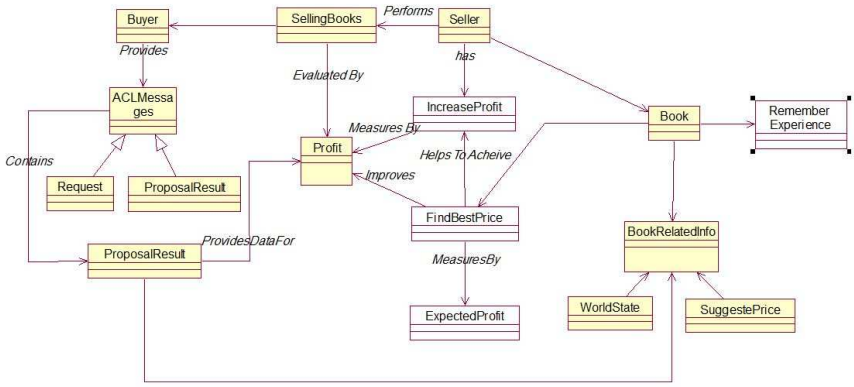


Fig. 2. Seller Agent Conceptual Model

To evaluate the applicability of our pattern and discovering how it can be realized during design and implementation, we compared our learning pattern participants with their instances on the case study and related design and implementation elements (such as agent, class, attribute and methods). The results show that, agent, task, environment, input, performance measure and knowledge are the meta-classes of the model, which there are design and implementation elements for them. These concepts have been highlighted as yellow meta-classes in figure 2. While goal, learning goal, learning element and learning measure are meta-classes which are used for understanding the domain of the application and they aren't instantiated as a design or implementation element. These classes are related to non-functional properties of learning and provide important information for designer that can help him/her for selecting appropriate algorithm for learner agent. They are also important for designing test cases of the agent and therefore can be useful for testers as other stakeholders of analysis artifacts. These concepts have been shown as white meta-classes in figure 2.

4 Conclusion and Further Work

In this paper, we introduced the first version of our analysis pattern for learning capability of software agents that can be used by software engineers during late requirements analysis. This pattern is defined in the terms of domain neutral meta-classes (and their relations) that can be identified as elements of conceptual modeling for analyzing and understanding the learning requirements of agents. According to this pattern, for analyzing the learning capability of an agent, its goals, tasks, learning goal, environment, data and feedback, knowledge, learning elements, learning measure and learning level should be considered. Conceptual models of learner agents in various domains can be constructed by applying the pattern on the application domain, which means these meta-models are instantiated in the domain. By using this pattern, the required information for understanding the learning requirement is provided during analysis phase. This will help the software analysts who are not expert in AI or learning to provide required information for designer to decide about learning algorithm and methods according to the application domain constraints.

Although our results provide some evidences about the applicability of our pattern in agent based systems, they also point out some limitations in our research, which we consider as part of our future work agenda to improve our pattern. We can mention the following among them; applying pattern on more complex case studies in various application domain, taking into account more resources of learning, evaluating pattern by other criterion, designing empirical case studies to evaluate the effectiveness of the pattern during software development process, investigating the usage of our pattern in various agent oriented methodologies.

Acknowledgments. The authors would like to thank Anna Perini for her comments on the early draft of the paper.

5 REFERENCES

[1] Zambonelli, F., Omicini, A.: Challenges and Research Directions in Agent-Oriented Software Engineering. J. Autonomous Agents and Multi-Agent Systems, Volume.9 No.3, 253—283 (2004)

[2] Ganek, A. G, Corbi, T. A.: The dawning of the autonomic computing era. J. IBM Systems 42(1),5--18,(2003).

[3] Sardinha, J. A. R. P., Garcia A. F., Milidiú R. L., Lucena C. J. P.: The Agent Learning Pattern. 4th Latin American Conference on Pattern Languages of Programming, SugarLoafPLoP'04, Fortaleza, Brazil,(2004)

[4] Alessandro F. Garcia, Uirá Kulesza, José Alberto R. P. Sardinha, Ruy L. Milidiú, Carlos J. P. Lucena.: The Learning Aspect Pattern. The 11th Conference on Pattern Languages of Programs (PLoP2004), (2004)

[5] Coad, P., D. North, M. Mayfield: Object Models: Strategies, Patterns, & Applications. Prentice Hall, Upper SaddleRiver, NJ. (1995)

[6] Fowler, M.: Analysis Patterns: Reusable Object Models. Addison-Wesley, Reading, MA. (1997)

Test Coverage Criteria for Agent Interaction Testing

Tim Miller[1], Lin Padgham[2], and John Thangarajah[2]

[1] Department of Computer Science and Software Engineering, University of Melbourne, Australia
[2] Department of Computer Science, RMIT University, Melbourne, Australia

Abstract. By the very definition of complex systems, complex behaviour emerges from the interactions between the individual parts. This emergent behaviour may be difficult or impossible to predict by analysing the parts. As a result, systematic and thorough testing of the interactions of complex systems, including multi-agent systems, is an important part of the verification and validation process. This paper defines two sets of test coverage criteria for multi-agent interaction testing. The first uses only the protocol specification, while the second considers also the plans that generate and receive the messages in the protocol. We describe how an existing debugging agent can be used as a test oracle for assessing correctness of a test, and how the Petri Net representation of the debugging agent can be annotated to support test coverage measurements. This work both specifies, and shows how to measure, the degree of thoroughness of a set of test cases. It also provides a basis for the future specification of test case input, designed to provide good coverage.

1 Introduction

Like other types of complex systems, the overall behaviour of multi-agent systems emerges from the interaction of their parts. Often, this emergent behaviour is difficult or even impossible to identify without running the system. This increased complexity makes verification and validation of these systems a non-trivial task. Furthermore, the fact that the behaviour cannot be accurately predicted implies that manual test case generation is unlikely to test the more complex behaviour. Automated test generation offers one solution to help with this problem.

Previous work on testing multi-agent systems [1, 2, 8, 10, 15, 16] has contributed to testing frameworks and automated test case generation. However, none have explicitly focused on testing interactions, the source of complexity in many systems. In many multi-agent methodologies, such as Prometheus, Tropos and OMaSE [3], interactions are captured via interaction protocols in design diagrams.

Our focus in this paper is on using protocol specifications, as well as information about how the interacting agents use these specifications, to define and measure systematic interaction testing. We also describe how correctness can be determined using the debugging agent of Poutakidis et al. [13]. Section 2 defines two sets of *test coverage criteria* for interaction testing, the first using

only the protocol specification, and the second including information about the plans involved in receiving and sending messages for a particular protocol. Section 3 describes the use of Poutakidis et al.'s "debugging agent" as a test oracle for determining whether a set of interacting agents is correctly following a valid protocol. Modifications to this debugging agent are made to automatically measure how well a test set achieves the coverage criteria. We finish with a discussion of relationships to previous work and a comment on future work.

2 Test Coverage Criteria

To measure the quality of a set of test cases, a criterion is necessary. Standard control-flow and data-flow criteria [9] that are defined for imperative programming languages are based on program statements and predicates, so are not directly applicable to agent interaction. However, many of the underlying ideas are valid. In this section, we define two sets of criteria based on the control-flow of interactions. This control-flow is extracted from the design models. The first set is based on the ordering of messages, which we obtain from protocol specifications. We refer to these as *protocol-based* criteria. The second set also considers the plans that send and receive the messages in protocols. We refer to these as *plan-based* criteria. We describe and compare each of these.

2.1 Protocol-based Coverage Criteria

Based on protocols specified in a standard protocol language such as AUML2 interaction diagrams, it is possible to construct a *protocol graph* that shows all possible orderings of messages[3]. Figure 1 shows the protocol graph corresponding to the FIPA Query Interaction Protocol [5].

The *conversation IDs* annotated to each message identify six conversations that have happened using this protocol, in which a conversation is a possible chaining of messages.

Criterion Definition Our coverage criteria are based on graph traversal of the protocol graph. For protocol coverage, we define three criteria:

Message coverage Every message in the protocol must be sent at least once.

Pairwise message coverage For every message, start node, and end node in the protocol, all directly proceeding messages/nodes must be executed after the first message/node at least once; that is, we must test every case in which one message can be followed by another.

Message path coverage Every possible interaction sequence permitted by the protocol must be executed at least once.

These three criteria correspond to node, arc, and path coverage of a graph. Figure 1 contains a minimal set of conversations that, if fully executed, achieve these criteria on the protocol graph.

[3] Our coverage criteria are then based on these orderings. We are not concerned with the content of messages, nor the time at which they are sent, only the relative ordering.

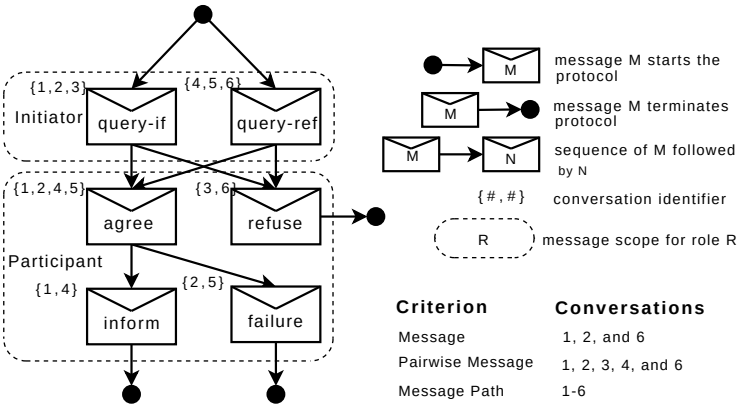


Fig. 1. A protocol graph for the FIPA Query interaction protocol specified, and the conversations required to achieve coverage criteria.

Achieving path coverage is sometimes not possible as a protocol may be defined as an infinitely iterative or recursive structure, leading to an infinite number of paths. Workarounds include achieving only non-cyclical path coverage, or using heuristics such as the 0-1-*many* rule, which specifies that we test only three of these paths: paths containing 0 loops, 1 loop, and more than one loop.

Coverage Measures Spillner [14] defines *coverage measures* for integration testing criteria. Coverage measures are defined as "*the ratio between the test cases [inputs] required for satisfying the criteria and those of these which have been executed*". These measures can be applied to test sets to determine how complete they are for a particular program.

The interaction coverage measures (IC) for our three protocol-based criteria are defined as follows:

$$IC_{protocol\_message} = \frac{\#\text{messages sent at least once}}{\#\text{totalmessages in protocol}}$$

$$IC_{pairwise\_message} = \frac{\#\text{arcs executed}}{\#\text{total arcs in protocol}}$$

$$IC_{message\_path} = \frac{\#\text{paths executed}}{\#\text{total message paths in protocol}}$$

As an example, in Figure 1 the set of conversations 1, 2, and 6 achieves 100% for protocol message coverage (6 messages that are all executed), 82% for pairwise message coverage (11 arcs, 9 arcs covered), and 50% for message path coverage (6 different paths, 3 paths covered).

Protocol-based coverage criteria are intuitively useful for interaction testing because they are strictly related to the interactions that can occur between the agents. However, purely message-based criteria do not consider the internal structure of the agents. For example, an agent may be able to send or receive the same message in many different plans. Consequently, we develop an additional set of coverage criteria that take into account the plans of the agents, and the relationship of messages to these plans.

2.2 Plan-based Coverage Criteria

We extract from the design artifacts, the information to build a *plan graph* for each protocol, of the kind shown in Figure 2. This graph represents the relationship between plans and messages for a particular protocol.

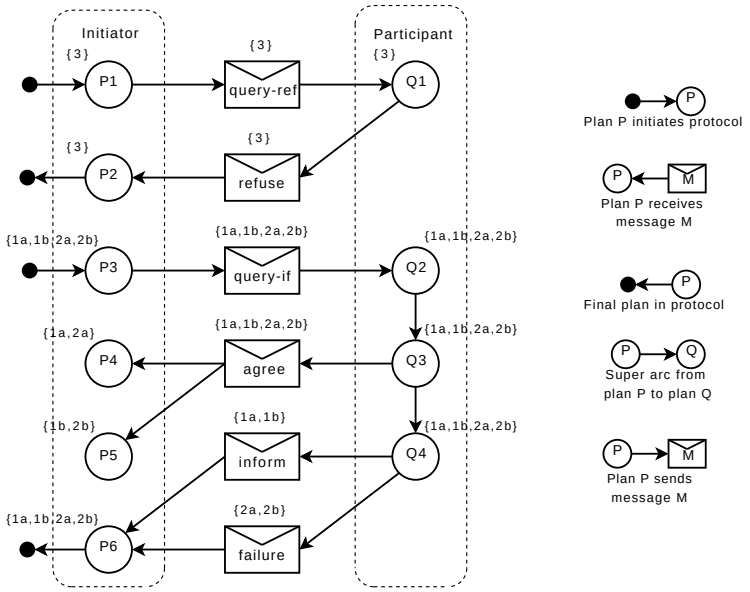
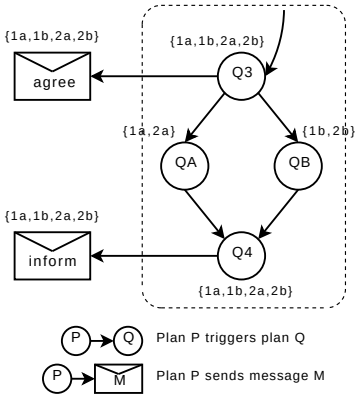


Fig. 2. A plan graph for the FIPA Query interaction protocol specified.

Plan graphs are built by extracting as nodes, those plans that send or receive any message in the protocol, and the messages themselves. In addition to the obvious send/receive links between plans and messages, we add a link between any two plans in the graph, which are connected by a chain (or multiple chains) of triggering links. We will call such links between plans *super-arcs* as they represent an entire plan structure. Figure 3 shows the internals of the super-arc between plan nodes $Q3$ and $Q4$ in Figure 2.

From Figure 2, one can see that the participant always agrees to a *query-if* request and always refuses a *query-ref*; so while the agents may follow the protocol, they do not use all parts of it. We also note that plans can send more than one message or receive more than one message, for example, plan $Q4$ sends both *inform* and *failure*.

Unlike other branches in the graph, the branch at $Q3$ is not a choice. Instead $Q3$ sends the message *agree*, and then triggers the plan $Q4$. For the purpose of test criteria, it is not necessary to model whether this is a choice or the ability to do more than one action because we need only measure whether the message was sent.



Criterion	Conversations
Message	1(a or b), 2(a or b), and 3
Plan	1a, 1b, and 3
Plan arc	1a, 2b, and 3
Pairwise internal path	1a, 1b, 2a, 2b, and 3
Plan path	1a, 1b, 2a, 2b, and 3

Fig. 3. Internal plan structure between plans $Q3$ and $Q4$ for the graph from Figure 2, and the conversations required to achieve coverage criteria for the entire protocol.

Criterion Definition We define a set of coverage criteria using plan graphs, in a similar way to those we defined on the protocol graph. We note how these correspond to criteria in standard (non-agent-oriented) integration testing [14].

Message coverage Every message in the plan graph is sent at least once. The analogous case in standard integration testing is ensuring that each method/function in the target component's interface is executed at least once.

Plan coverage Every plan that sends or receives a message in the protocol is executed at least once. The analogous case in integration testing is ensuring that each method in the program that calls a method in the target component's interface is executed.

Plan arc coverage Every occurrence of a message being sent by a plan and every occurrence of a plan being triggered (by a message, an event (a start node), or another plan) is executed at least once. This is different from plan coverage because a plan may be able to send more than one message (e.g. plan $Q4$ sending *inform* and *failure* in Figure 2). The analogous case in integration testing is ensuring that every call made to every method in the target component's interface is tested.

Pairwise internal path coverage Every possible path, including paths in super arcs, between two pairwise messages, or between a first/last message in a protocol and its corresponding start/end node is executed at least once. This ensures that all paths that could be used to produce a particular message in the protocol, are tested. The analogous case in integration testing is ensuring that every path between two method calls from the target component's interface is executed. Note that pairwise messages cannot be determined from the plan graph, but must be determined from the protocol specification or protocol graph. For example, in Figure 2, one cannot determine that *agree* is sent directly before *inform*.

Plan path coverage Every possible path through the structure induced by expanding super-arcs within the plan graph is executed at least once. The anal-

ogous case in integration testing is ensuring that every possible sequence of calls to every method in the target component's interface is tested. Even without the expansion of super-arcs this differs from message path coverage defined on the protocol graph, in that it addresses the case where the same message may be sent from, or received by, two different plans. (e.g. plan $P4$ and $P5$ receiving *agree* in Figure 2).

Again, some of the above criteria correspond to graph coverage criteria. Message and plan coverage combined correspond to node coverage. Plan arc coverage, and plan path coverage correspond to arc, and path coverage respectively. Pairwise internal path coverage corresponds to path coverage between plan nodes within a super arc, combined with arc coverage on the other arcs of the graph.

To illustrate, Figure 3 contains a minimal test sets that, when fully executed, achieve each criteria, using the plan graph from Figures 2 and 3.

Coverage Measures We define coverage measures for these criteria in the same way as the protocol-based criteria: the ratio of executed nodes/arc/paths to the total number of nodes/arc/paths.

For example, in Figures 2 and 4, the set of conversations 1a, 2b, and 3 achieves 100% coverage for message coverage (6/6), plan coverage (10/10), and plan arc coverage (19/19), 85% for pairwise internal path coverage (11/13), and 55.5% for plan path coverage (5/7).

2.3 Comparison of Coverage Criteria

To compare these criteria, we are interested in any subsumption relationships between them. Test criterion A *subsumes* test criterion B if and only if any test set that achieves 100% coverage on criterion A also achieves 100% coverage on criterion B.

Figure 4 shows the subsumption relationship between our criteria. We know from graph theory that path coverage subsumes arc coverage, and arc coverage subsumes node coverage. This directly gives us the subsumption relation between the different protocol graph criteria (message path subsumes pairwise message, and pairwise message subsumes message).

In the plan graph, plan path subsumes plan arc, and plan arc subsumes message/plan coverage, directly from graph theory. Although neither message nor plan coverage are equivalent to node coverage, both are subsumed by it. Plan coverage does not subsume message coverage, because plans can send and receive multiple messages. For example, plan coverage of Figure 2 can be achieved by executing plans $Q4$ and $P6$ once each, which means either *inform* or *failure* will not be sent. There is also an additional coverage metric, pairwise internal path coverage, which sits between plan path coverage and plan arc coverage. The argument for this is straightforward: by definition it subsumes arc coverage, and if every path, including every path internal to a super arc, is executed, then every arc plus all super-arc paths must also be executed.

To compare the criteria for the two different types of graph, we make the assumption that all criteria are feasible. For example, in plan message coverage,

we assume that the participating agents are programmed such that every message in a protocol is able to be sent by these agents. Otherwise, 100% coverage is not achievable. It is not uncommon for this assumption to be false, particularly when pre-existing protocols are used. For example, a developer using a third-party protocol may choose not to use some messages defined in a protocol.

If this assumption is relaxed, the result is simply that there is no subsumption relation between any of the criteria[4]. With this assumption of feasibility we can establish some relationships between the criteria based on the two different graphs. Firstly, we note that the two types of message coverage are equivalent. That is, they both require test cases that send every message in the protocol. The next relation is that pairwise internal path coverage subsumes pairwise message coverage. Pairwise internal path coverage is defined as executing all paths (including super arcs) between all pairwise messages, therefore, it trivially subsumes pairwise message coverage. Finally, we have that plan path coverage subsumes message path coverage. With our assumption of feasibility, this subsumption relation holds because if there is a path defined by the protocol, there must be a path in the plan graph that executes it. If all paths through the plan graph are executed, then this implies all paths in the protocol graph must also be executed.

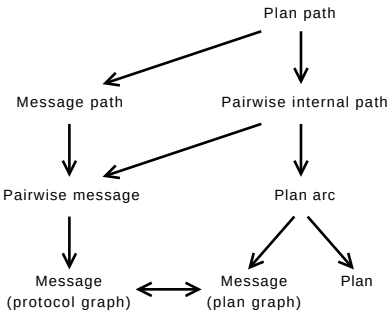


Fig. 4. The *subsumes* relation between the protocol-based and plan-based coverage.

We argue that the combination of message path coverage and pairwise internal path coverage is a minimum testing level to aim for in rigorous interaction testing. It tests the various plan combinations that may be used in moving from receipt of a message, to the production of the next message in the protocol, and also tests every possible conversation. Although there is some amount of exponential growth, this is likely to be substantially more limited than that required for testing all paths in the plan graph.

3 Measuring Correctness and Coverage Using a Debugging Agent

The model-based measure of correct behaviour of agent interaction is primarily whether the agents follow the specified interaction protocols. While the coverage measures we have defined can tell us how thoroughly a given set of test cases actually exercises the program under test, we require some way of knowing whether the agents interact as specified. To establish this we use the work of

[4] This can be demonstrated by the examples in Figures 1 and 2: the agents are programmed such that the sequence $\langle \textit{query-ref} \rightarrow \textit{agree} \rangle$ is infeasible, therefore, pairwise message coverage is not achievable on the protocol graph, but pairwise internal path coverage is achievable on the plan graph.

Poutakidis et al. [13] on debugging agent interactions. The monitor that is used in that work for detecting bugs, can equally well be used as a *test oracle*.

The IEEE Standard Glossary of Software Engineering Terminology [6] defines a test oracle as: "*any means of determining whether a system or component's behaviour is consistent with its specification.*"

In Poutakidis et al.'s work, the agent platform is modified so that the debugging agent receives copies of all messages sent within the system. This debugging agent then raises an alert if a sent message does not follow one of the specified protocols, or if a protocol does not reach a specified end state. These are the two possible errors that can arise with respect to the agent interactions.

We use the infrastructure of Poutakidis et al. to collect information regarding our protocol graph interaction coverage criteria. This information can be collected by an independent observer. For the plan-graph coverage criteria, information must be known about the inner details of the participants. In current work, we are adapting Zhang et al.'s automated unit test framework [16] to measure plan-graph coverage criteria.

3.1 Petri-Net Representation for Protocols

Poutakidis et al. systematically translate AUML protocol specifications into Petri Nets, and executing these as agents interact, are able to ascertain whether the interaction is following a specified protocol.

A Petri Net is a bipartite graph containing two types of nodes: *places* and *transitions*. Places are represented with circles, and transitions are represented with rectangles (see Figure 5). Arcs connect transitions to places. The execution semantics of Petri Nets specifies that *tokens* can be located at places. A transition can be *fired* if all incoming places contain a token and the outgoing place is empty; when the transition fires, a token is placed at all outgoing places.

Poutakidis et al. define Petri Nets with two kinds of places: state places and message places. State places represent the state prior to a given message being received, or end states. When a Petri Net instance is initialised by the debugger, it has a token placed on its relevant message and state places. At each cycle all Petri Net instances are fired to completion, and then retained until the next cycle, when a token is added to the message place in the relevant Petri Nets. Poutakidis et al. define mappings from protocols to Petri Nets to model the possible protocol executions.

Figure 5(a) shows the Petri Net for the FIPA query protocol. When a *query-if* message arrives, this is identified as a start message for this protocol, and a new Petri Net instance is created. A token is placed in the *query-if* message place, and the corresponding state place. The Petri Net is then executed allowing the transition to fire producing a token on the outgoing state-place, P, as in Figure 5(b). The Petri Net is now in a state where, when a token is placed on either the *agree* or *refuse* message place, it can fire the relevant transition, producing a token in either R or T.

The debugging agent contains a copy of every protocol (and its corresponding Petri Net) used in the system. Each conversation held between a set of agents must contain a *conversation ID* to allow placement of a message into the correct

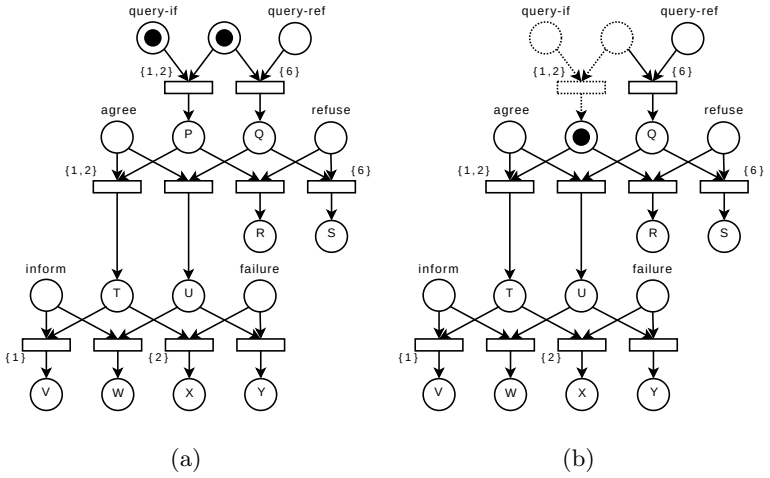


Fig. 5. An example of a Petri Net transformation.

Petri Net instance as there may be concurrent conversations. This requirement is supported by the FIPA standard for agent communication [4].

Each time the debugging agent receives a copy of a message, it first confirms whether the conversation ID corresponds to a current conversation. If not, it creates a Petri Net for *all* protocols that contain the received message as their initial message, initialises these appropriately, (as in Figure 5(a)). This is necessary because it has no way of knowing which protocol the sending agent is using if more than one protocol has that message as the initial message. The agent maintains a set of Petri Net instances for each conversation, until it becomes clear by a process of elimination, which protocol is being used.

If the message corresponds to an existing conversation, the debugging agent places the message in the appropriate message-place of all the Petri Nets in the set for the conversation. Those that do not have an appropriate place, or where the message place does not enable a transition, are removed from the set, because it is evident the conversation is not following this protocol. If the set becomes empty, then this indicates a fault: the agents are not following any known protocol. For example, if the Petri Net is in the state shown in Figure 5(b), and the debugging agent received the message *agree*, then this transition could fire. Alternatively, if the message was *refuse*, the transition could not fire because there would be no token at Q. This indicates a fault.

The process continues until the conversation is deemed to have terminated. If there are tokens remaining in any non-terminal places, this indicates a fault, because the conversation did not follow the protocol to completion.

3.2 Measuring Coverage Using Petri Nets

We adapt Poutakidis et al.'s debugging agent to measure the coverage corresponding to the protocol-based criteria defined in Section 2. First, we modify

the oracle such that, when a transition is fired, the transition is annotated with the conversation ID, such as in Figure 5(a).This records all of the conversations that take place using a particular protocol. In this example, conversations 1, 2, and 6 from Figure 1 have been executed, and the appropriate transitions have been annotated.

To measure message coverage, we analyse each message place in the Petri Net and determine if at least one of its outgoing transitions has been fired (is annotated). If so, the message has been sent. We can then use the coverage measure definition from Section 2.1 to measure coverage.

To measure pairwise message coverage, we analyse each place in the Petri Net that represents an intermediate state; that is, all non-message-places between two transitions. If the incoming transition and at least one outgoing transition share at least one conversation ID, then this pair was executed. For example, the place P contains one incoming transition and two outgoing transitions. The incoming arc and the left outgoing transition both contain the conversation IDs 1 and 2, so this pair was executed in sequence. To show this is valid, we note that the unfolding rules specified by Poutakidis et al. [13] result in a graph such that any two places are linked by at most one path. As a result, each intermediate place in a Petri Net must have at least one input and one output transition, and all pairwise messages in a protocol are connected by exactly one such message place in the Petri Net representation. If the incoming and outgoing message place share a conversation ID, then the pair of messages must have been executed.

Finally, to measure message path coverage, we take each terminal place, and determine if the incoming transition to that place was fired; that is, contains at least one conversation ID. To demonstrate validity of this, we again note that Poutakidis et al.'s unfolding rules result in a graph such that any two places are linked by at most one path. Therefore, the final transition is unique to a path, so if this transition has been fired, the entire path must have been executed.

3.3 Measuring Coverage for Concurrent Conversations

To monitor multiple conversations over a single protocol, Poutakidis et al. create multiple instances of the same Petri Net. Using a single instance is not suitable because, upon testing to see if a message is valid, the Petri Net may be in a configuration such that the message is valid for another conversation, but not the current one. Creating multiple instances of a Petri Net is suitable for monitoring interactions, however, to measure coverage, the coverage information is spread over multiple Petri Nets for a single protocol.

One solution is to collate all information from all Petri Nets after a test suite has been executed. However, this is somewhat inefficient and cumbersome.

A more elegant solution is to adapt Poutakidis et al.'s solution to use *coloured Petri Nets* [7]. Coloured Petri Nets extend Petri Nets by (among other things) allowing tokens to carry a *value*. A transition can be fired only if all incoming places contain a token with the same shared value.

To handle concurrent conversations, each protocol corresponds to a single Petri Net in the test oracle. Rather than creating multiple copies for multiple conversations, tokens are given values corresponding to the conversation ID. Using this, the test oracle receives a message containing a conversation ID, and can

determine at which place the correct token resides. From here, it can determine whether the message is valid.

Taking this approach, tokens remain at the terminal places after conversations have terminated. Therefore, measuring path coverage is as simple as counting the number of terminal places that contain at least one token, and dividing this by the total number of terminal places.

4 Discussion and Conclusion

Due to the complex emergent behaviour that results from agents interacting with each other, testing these interactions is an important part of the verification and validation process. The fact that emergent behaviour in complex systems is often difficult or impossible to identify without running these systems implies that using human test engineers to generate test cases manually is not sufficient, and automated test case generation techniques are required.

There has been recent work on automating test case generation such as the Unit test framework of Zhang et al. [16] already mentioned in this work, and the eCAT system associated with Tropos [10–12]. eCAT is a testing tool that automates test case generation and execution. There are 4 test generation techniques employed in eCAT: *goal-oriented*, which is manual test generation using goal diagrams; *ontology-based*, where test cases are derived automatically from the specification of the agent interaction ontology; *random*, where values for test cases are randomly generated; and *evolutionary mutation*, where genetic algorithms generate test cases measured by the quality goals of the system. Our approach to testing correctness, and measuring thoroughness could complement any of these test case generation techniques.

Whether test cases are generated automatically or manually, it is important to have a measure of the quality of the set of test cases. This paper has provided criteria by which to measure this, showing the subsumption relationships between these criteria, and discussing which we would practically aim for. We suggest that testing all paths through the protocol, combined with all plan paths between two messages achieves a high level of coverage, and is likely to be more feasible than plan path coverage, which subsumes both of these criteria. This paper has also shown how to collect these measurements as part of the testing process. We consider that these coverage definitions provide a sound basis for guiding test case generation where test cases are designed to give good coverage.

Low et al. [8] also consider test coverage criteria for BDI agents. They derive two types of control-flow graphs: one with nodes representing plans and arcs representing messages or other events that trigger plans; and one with nodes representing statements within plans and arcs representing control-flow between statements (a standard control-flow graph). Several coverage criteria are defined, based on node, arc, and path coverage, as well as some based on the success or failure of executing statements and plans. However, Low et al.'s work builds graphs over the entire program, and thus does not facilitate the modular and focused testing based on specific interaction protocols.

Low et al.'s coverage criteria relate to ours. Their plan graph is similar to our plan graph, except that they consider plans that are not related to interaction. As a result, their coverage criteria subsume ours; for example, their plan

path coverage subsumes our plan path coverage. However, their criteria do not consider pairwise messages, as they do not focus on interaction protocols. Low et al. do not define specific coverage measures or how to calculate them, nor do they discuss test oracles.

The work in this paper is one step towards a larger goal: model-based automated testing for multi-agent systems. Future work will define methods for automatically deriving test cases from design artifacts. With respect to interaction testing, we will attempt to automatically generate complete test suites that achieve message path coverage combined with pairwise internal path coverage, using design documents as the models.

References

1. G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci. Multi-Agent Systems Implementation and Testing. In *the Fourth International Symposium: From Agent Theory to Agent Implementation*, Vienna, April 14-16 2004.
2. R. Coelho, U. Kulesza, A. von Staa, and C. Lucena. Unit testing in multi-agent systems using mock agents and aspects. In *Proc. of the 2006 Intl. Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, pages 83–90, 2006.
3. S. DeLoach, L. Padgham, A. Perini, A. Susi, and J. Thangarajah. Using three AOSE toolkits to develop a sample design. *International Journal of Agent-Oriented Software Engineering*, 3(4):416–476, 2009.
4. FIPA. FIPA ACL message structure specification. Standard SC00061G, Foundation for Intelligent Physical Agents, December 2002.
5. FIPA. FIPA query interaction protocol specification. Standard SC00027H, Foundation for Intelligent Physical Agents, December 2003.
6. IEEE. IEEE standard glossary of software engineering terminology. Technical Report 610.12-1990, Institute of Electrical and Electronic Engineers, 1990.
7. K. Jensen. *Coloured Petri Nets*. Springer Verlag, 1997.
8. C. Low, T. Y. Chen, and R. Ronnquist. Automated test case generation for BDI agents. *Autonomous Agents and Multi-Agent Systems*, 2(4):311–332, 1999.
9. Glenford J. Myers. *The Art of Software Testing*. Wiley, New York, 1979.
10. C. Nguyen, A. Perini, and P. Tonella. Automated continuous testing of multi-agent systems. In *Fifth European Workshop on Multi-Agent Systems*, Hammamet, Tunisia, December 2007.
11. C. Nguyen, A. Perini, and P. Tonella. eCAT: a tool for automating test case generation and execution in testing multi-agent systems (demo paper). In *Proceedings of AAMAS-08*, pages 1669–1670, Estoril, Portugal, 2008.
12. C. Nguyen, A. Perini, and P. Tonella. Ontology-based test generation for multi-agent systems. In *Proceedings of AAMAS-08*, pages 1315–1320, 2008.
13. D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of AAMAS-02*, pages 960–967, 2002.
14. A. Spillner. Test criteria and coverage measures for software integration testing. *Software Quality Journal*, 4(4):275–286, 1995.
15. A. Tiryaki, S. Öztuna, O. Dikenelli, and R. Cenk Erdur. SUNIT: A unit testing framework for test driven development of multi-agent systems. In *Agent-Oriented Software Engineering VII*, volume 4405 of *LNCS*, pages 156–173, 2006.
16. Z. Zhang, J. Thangarajah, and L. Padgham. Automated unit testing for agent systems. In *2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering*, pages 10–18, Spain, July 2007.

Model-Driven Agents Development with ASEME

Nikolaos Spanoudakis[1,2] and Pavlos Moraitis[2]

[1]Technical University of Crete, Dept of Sciences, University Campus, 73100 Chania, Greece
nikos@science.tuc.gr
[2]Laboratory of Informatics Paris Descartes (LIPADE), Paris Descartes University,
45 rue des Saints-Pères, 75270 Paris Cedex 06, France
{Nikolaos.Spanoudakis, pavlos}@mi.parisdescartes.fr

Abstract. This paper shows how an AOSE methodology, the Agent Systems Engineering Methodology (ASEME), uses state of the art technologies from the Model-Driven Engineering (MDE) domain. We present the Agent Modeling Language (AMOLA) Metamodels and the model transformation tools that we developed and discuss our choices. Then, we compare ASEME with a set of existing AOSE methodologies.

Keywords: Model Driven Engineering, Agent Oriented Software Engineering

1 Introduction

During the last years, there has been a growth of interest in the potential of agent technology in the context of software engineering. A new trend in the Agent Oriented Software Engineering (AOSE) field is that of converging towards the Model-Driven Engineering (MDE) paradigm. Thus, a lot of well known AOSE methodologies propose methods and tools for automating models transformations, such as Tropos [18] and INGENIAS [4], but this is done only for some of the software development phases.

This paper aims to show for the first time how the principles of MDE can be used throughout all the software development phases and how the AOSE community can use three different types of transformations in order to produce new models based on previous models. This approach has been used by the Agent Systems Engineering Methodology[1] (ASEME) [21, 22] and shows how an agent-based system can be incrementally modeled adding more information at each step using the appropriate type of model.

ASEME offers some unique characteristics regarding the used MDE approach. It covers all the classic software development phases (from requirements to implementation) and the transition of one phase to another is done through model transformations. It employs three transformation types, i.e. model to model (M2M),

[1] From the ASEME web site the interested reader can download the tools and metamodels used in this paper, URL: http://www.amcl.tuc.gr/aseme

text to model (T2M) and model to text (M2T). Thus, the analysts/engineers and developers just enrich the models of each phase with information, gradually leading to implementation. Moreover, the design phase model of ASEME is a statechart [7], a modeling paradigm well known to engineers, which can be instantiated using a variety of programming languages or an agent-oriented framework.

This paper presents the ASEME process showing the models transformations between the different development phases. The models that are used by ASEME are defined by the Agent Modeling Language (AMOLA, a first version is presented in [23]). Section two provides a background on metamodeling and models transformation followed by the definition of the AMOLA metamodels in section three. The ASEME MDE process is presented in section four discussing the used transformation tools. An overview of the related work and a brief discussion of our findings conclude the paper in section five.

2 Metamodeling and Models Transformation

Model driven engineering relies heavily on model transformation [20]. Model transformation is the process of transforming a model to another model. The requirements for achieving the transformation are the existence of metamodels of the models in question and a transformation language in which to write the rules for transforming the elements of one metamodel to those of another metamodel.

In the software engineering domain a *model* is an abstraction of a software system (or part of it) and a *metamodel* is another abstraction, defining the properties of the model itself. However, even a metamodel is itself a model. In the context of model engineering there is yet another level of abstraction, the *metametamodel*, which is defined as a model that conforms to itself [10].

There are four types of model transformation techniques [12]:

- **Model to Model (M2M)** transformation. This kind of transformation is used for transforming a type of graphical model to another type of graphical model. A M2M transformation is based on the source and target metamodels and defines the transformations of elements of the source model to elements of the target model.
- **Text to Model (T2M)** transformation. This kind of transformation is used for transforming a textual representation to a graphical model. The textual representation must adhere to a language syntax definition usually using BNF. The graphical model must have a metamodel. Then, a transformation of the text to a graphical model can be defined.
- **Model to Text (M2T)** transformations. Such transformations are used for transforming a visual representation to code (code is text). Again, the syntax of the target language must be defined along with the metamodel of the graphical model.
- **Text to Text (T2T)** transformations. Such transformations are used for transforming a textual representation to another textual representation. This is usually the case when a program written for a specific programming language is transformed to a program in another programming language (e.g. a compiler).

In the heart of the model transformation procedure is the Eclipse Modeling Framework (EMF, [2]). Ecore [2] is EMF's model of a model (metamodel). It functions as a metametamodel and it is used for constructing metamodels. It defines that a model is composed of instances of the *EClass* type, which can have attributes (instances of the *EAttribute* type) or reference other EClass instances (through the *EReference* type). Finally, EAttributes can be of various *EDataType* instances (such are integers, strings, real numbers, etc).

A similar technology, the Meta-Object Facility (MOF), is an OMG standard [14] for representing metamodels and manipulating them. MOF is older than EMF and it influenced its design. However, the EMF meta-model is simpler than the MOF meta-model in terms of its concepts, properties and containment structure, thus, the mapping of EMF's concepts into MOF's concepts is relatively straightforward and is mostly 1-to-1 translations [5]. EMF is also used today by a large open source community becoming a de facto standard in MDE.

3 The AMOLA Metamodels

System Actor Goal model (SAG)

The SAG model is a subset of the Actor model of the Tropos ecore model [27]. Tropos is, on one hand, one of the very few AOSE methodologies that deal with requirements analysis, and, on the other hand it borrows successful practices from the general software engineering discipline. This is why we have been inspired by Tropos. The reason for not using the Tropos diagrams as they are is that they provide more concepts than the ones used by AMOLA as they are also used for system analysis. However, as we will show later, AMOLA defines more well-suited diagrams for system analysis. Thus, the AMOLA System Actors Goals diagram is the one that appears in Figure 1(a) employing the *Actor* and *Goal* concepts. The actor references his goals using the *EReference my_goal*, while the *Goal* references a unique *depender* and zero or more *dependees*. The reader should notice the choice to add the *requirements EAttribute* of *Goal* where the requirements per goal information is stored.

Use case model (SUC)

In the analysis phase, the analyst needs to start capturing the functionality behind the system under development. In order to do that he needs to start thinking not in terms of goal but in terms of what will the system need to do and who are the involved actors in each activity. The use case diagram helps to visualize the system including its interaction with external entities, be they humans or other systems. It is well-known by software engineers as it is part of the Unified Modeling Language (UML).

In AMOLA no new elements are needed other than those proposed by UML, however, the semantics change. Firstly, the actor "enters" the system and assumes a role. *Agents* are modeled as roles, either within the system box (for the agents that are to be developed) or outside the system box (for existing agents in the environment).

Human actors are represented as roles outside the system box (like in traditional UML use case diagrams). This approach aims to show the concept that we are modeling artificial agents interacting with other artificial agents or human agents. Secondly, the different use cases must be directly related to at least one artificial agent role.

The SUC metamodel containing the concepts used by AMOLA is presented in Figure 1(b). The concept *UseCase* has been defined that can include and be included by other *UseCase* concepts. It interacts with one or more roles, which can be Human roles (HumanRole) or Agent roles (SystemRole).

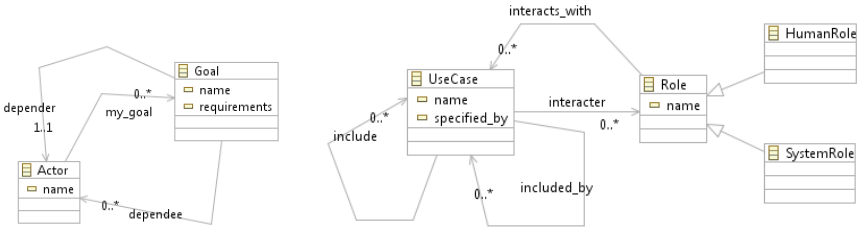


Fig. 1. The AMOLA SAG (a) and SUC (b) metamodels

Role model (SRM)

An important concept in AOSE is the role. An agent is assumed to undertake one or many roles in his lifetime. The role is associated with activities and this is one of the main differences with traditional software engineering, the fact that the activity is no longer associated with the system, rather with the role. Moreover, after defining the capabilities of the agents and decomposing them to simple activities in the SUC model we need to define the dynamic composition of these activities by each role so that he achieves his goals. Thus, we defined the SRM model based on the Gaia Role model [29]. Gaia defines the liveness formula operators that allow the composition of formulas depicting the role's dynamic behavior. However, we needed to change the role model of Gaia in order to accommodate the integration in an agent's role the incorporation of complex agent interaction protocols (within which an agent can assume more than one roles even at the same time), a weakness of the Gaia methodology. The AMOLA SRM metamodel is presented in Figure 2(a). The SRM metamodel defines the concept *Role* that references the concepts:

- *Activity*, that refers to a simple activity with two attributes, name (its name) and functionality (the description of what this activity does),
- *Capability* that refers to groups of activities (to which it refers) achieving a high level goal, and,
- *Protocol*. The protocol attributes *name* and *participant* refer to the relevant items in the Agent Interactions Protocol (AIP) model. This model is not detailed here-in. It is used for identifying the roles that participate in a protocol, their activities within the protocol and the rules for engaging (for more details consult [24]).

The *Role* concept also has the *name* and *liveness* attributes (the first is the role name and the second its liveness formula). The reader should note the *functionality*

attribute of the *Activity* concept which is used to associate the activity to a generic functionality. For example, the "get weather information" activity can be related to the "web service invocation" functionality (see [23], [25]).

Intra-agent control model (IAC)

In order to represent system designs, AMOLA is based on statecharts, a well-known and general language and does not make any assumptions on the ontology, communication model, reasoning process or the mental attitudes (e.g. belief-desire-intentions) of the agents, giving this freedom to the designer. Other methodologies impose (like Prometheus or INGENIAS [8]), or strongly imply (like Tropos [8]) the agent mental models. Of course, there are some developers who want to have all these things ready for them, but there are others who want to use different agent paradigms according to their expertise. For example, one can use AMOLA for defining Belief-Desire-Intentions based agents, while another for defining procedural agents [21].

The inspiration for defining the IAC metamodel mainly came from the UML statechart definition. Aiming to define the statechart using the AMOLA definition of statechart [26], the IAC metamodel differs significantly from the UML statechart. However, a UML statechart can be transformed to an IAC statechart although some elements would be difficult to define (UML does not cater for transition expressions and association of variables to nodes and uses statecharts to define a single object's behaviour). Thus, the IAC metamodel, which is presented in Figure 2(b), defines a *Model* concept that has *nodes*, *transitions* and *variables* EReferences. Note that it also has a *name* EAttribute. The latter is used to define the namespace of the IAC model. The namespace should follow the Java or C# modern package namespace format. The nodes contain the following attributes:

- *name*. The name of the node,
- *type*. The type of the node, corresponding to the type of state in a statechart, typically one of AND, OR, BASIC, START, END (see [7]),
- *label*. The node's label, and
- *activity*. The activity related to the node.

Nodes also refer to *variables*. The Variable EClass has the attributes *name* and *type* (e.g. the variable with *name* "count" has *type* "integer"). The next concept defined in this metamodel is that of *Transition*, which has four attributes:

- *name*, usually in the form <source node label>TO<target node label>
- *TE*, the transition expression. This expression contains the conditions and events that make the transition possible. Through the transition expressions (TEs) the modeler defines the control information in the IAC. TEs can use concepts from an ontology as variables. Moreover, the receipt or transmission of an inter-agent message can be used (in the case of agent interaction protocols). For the formal definition of the TE and some examples see [21] or [24].
- *source*, the source node, and,
- *target*, the target node.

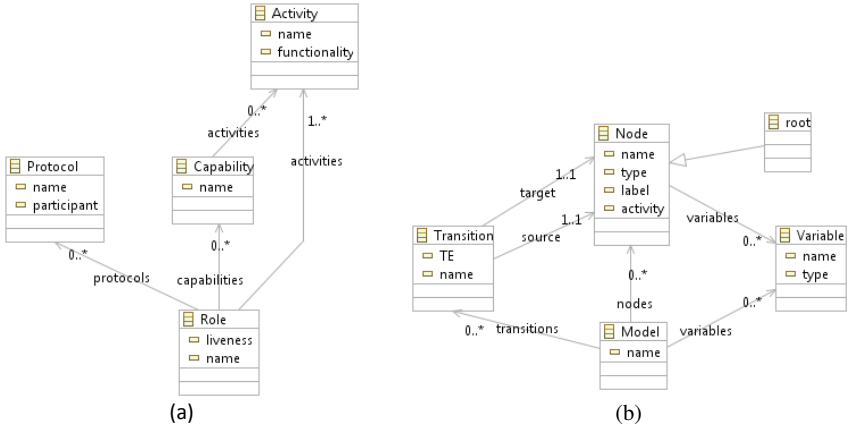


Fig. 2. The AMOLA SRM (a) and IAC (b) metamodels.

4 The ASEME Model-Driven Process and Tools

ASEME is described in detail in [21]. It is a complete process incorporating all the traditional software engineering methodology phases, however, using the SPEM 2.0 process metamodel [17] it can be modified to provide an agile process. Figure 3, a screenshot from the EPF[2] modeling tool, shows on the left side the ASEME method library and its various properties. From top to bottom the most important are the:

- *Work Product Kinds*, we have defined two product kinds, models (graphical models, e.g. SAG, SUC, etc) and text (textual representation, e.g. a computer program).
- *Role sets*, where the different human actors implicated in the software development process are identified.
- *Tools*, the various tools used in the process, in this case the transformation tools.
- *Processes*, can be *delivery processes*, which provide the project manager with an initial project template, showing the project milestones with the work products to be delivered and needed resources, or *capability patterns* that allow project managers to use one or more method libraries to compose their project-specific process.

In Figure 3, the reader can see two defined capability patterns, the first named ASEME and containing the six software development phases, and a more compact one, the ASEME MDE process where the model-driven development process for a single agent system is depicted. This process shows the nine tasks needed for developing an agent-based system:

[2] The Eclipse Process Framework (EPF) aims at producing a customizable software process engineering framework. URL: http://www.eclipse.org/epf/

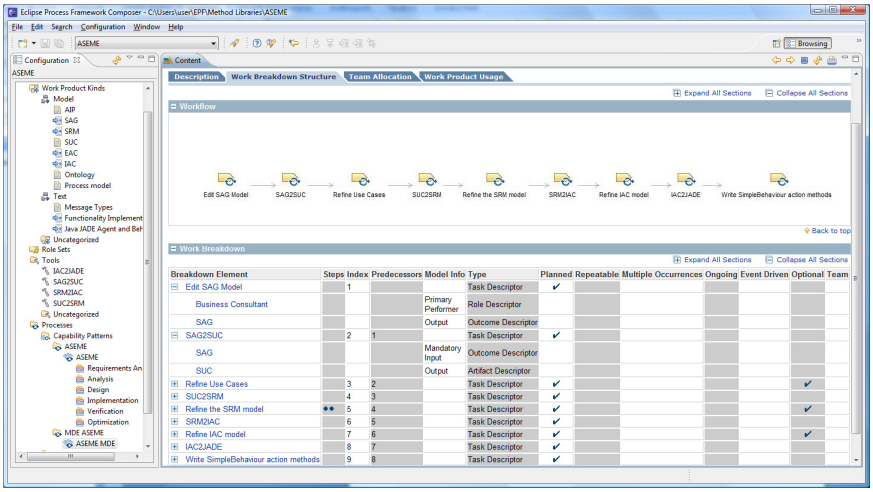


Fig. 3. The ASEME MDE Process

1. *Edit SAG model*. The business consultant of the software development firm identifies the actors involved in the system to be along with their goals.
2. *SAG2SUC*. An automated task, as the reader can see in the figure this task has only a mandatory input model (SAG) and an output model (SUC). It creates an initial SUC model based on the previously created SAG model.
3. *Refine Use Cases*. The analyst works on the SUC model and refines the general use cases using the include relationship. He/she also identifies which actors will be implemented defining them as human or artificial agent actors. The overall system design is enriched by identifying the tasks that have to be carried out by the actors.
4. *SUC2SRM*. An automated task, it has only a mandatory input model (SUC) and an output model (SRM). It creates an initial SRM model based on the previously created SUC model.
5. *Refine the SRM model*. The analyst works on the SRM model by defining the liveness formulas that will describe the dynamic compilation of the previously identified tasks.
6. *SRM2IAC*. An automated task, it has only a mandatory input model (SRM) and an output model (IAC). It creates multiple initial IAC models based on the previously created SRM model, one for each role.
7. *Refine the IAC model*. The designer works on each IAC model by defining the conditions and/or events that will enable the transitions from one task to the other.
8. *IAC2JADE*. An automated task, it has only a mandatory input model (IAC) and an output model (Java JADE Agent and Behaviours code). It creates a JADE Agent class and multiple JADE Behaviour classes for each IAC model.
9. *Write SimpleBehaviour action methods*. The programmer writes code only for the JADE *SimpleBehaviour* class descendants' *action* methods.

ASEME M2M Transformation Tools (SAG2SUC and SUC2SRM)

For model to model (M2M) transformation the Atlas Transformation Language [11] was used (ATL). Another alternative to ATL would be the Query-View Transformation (QVT) language [16], however, ATL was better documented on the internet with a user guide and examples, while the only resource located for QVT was a presentation. Therefore, and as the requirements of both languages (ATL and QVT) are the same the decision was to choose the better documented one. Such transformations are the SAG2SUC and SUC2SRM.

The ATL rules for the SAG2SUC transformation are presented in Figure 4. At the top of the right window, the IN and OUT metamodels are defined followed by rules that have an input model concept instance and one or more output concept model instances. The first rule (*Goal2UseCase*) takes as input a SAG Goal concept and creates a SUC UseCase concept copying its properties. The ATL is declarative and has catered for the cases that a concept references another. The *depender* and *dependee* references of a SAG Goal are both transformed to *participator* references of the SUC UseCase. The ATL engine searches the rules to find one that transforms the types of the EReference (i.e. the SAG Actor concepts to a SUC Role). It finds the second rule (*Actor2Role*) and fires it, thus creating the EReference type objects and completing the first rule firing. At the left hand side of Figure 4 the reader can see the files relevant to this transformation: a) the *SAG.ecore* and *SUC.ecore* metamodel files, b) the *SAG2SUC.atl* rules file, c) the *SAGModel.xmi* file containing the SAG model in XML format and d) the *SUCModelInitial.xmi* file containing the automatically derived initial SUC model.

ASEME T2M Transformation Tool (SRM2IAC)

The trick in text to model transformations is to define the meta-model of the text to be transformed. This can be done in the form of an EBNF syntax (for languages with a grammar) or through string manipulation. Efftinge and Völter [3] presented the xtext framework in the context of the Eclipse Modeling Project (EMP[3]). According to their work, an xText grammar is a collection of rules. Each rule is described using sequences of tokens. Tokens either reference another rule or one of the built-in tokens (e.g. STRING, ID, LINE, INT). A rule results in a meta type, the tokens used in the rule are mapped to properties of that type. xText is used to automatically derive the meta model from the grammar. Then a textual representation of a model following this grammar can be parsed and the meta-model is automatically generated.

Rose et al. [19] described an implementation of the Human-Usable Textual Notation (HUTN) specification of OMG [15] using Epsilon, which is a suite of tools for MDE. OMG created HUTN aiming to offer three main benefits to MDE: a) a generic specification that can provide a concrete HUTN language for any model, b) the HUTN languages to be fully automated both for production and parsing, and, c) the HUTN languages to conform to human-usability criteria. The HUTN implementation automates the transformation process by eliminating the need for a

[3] The Eclipse Modeling Project provides a unified set of modeling frameworks, tooling, and standards implementations, URL: http://www.eclipse.org/modeling/

grammar specification by auto defining it accepting as input the relevant EMF meta-model. This is the main reason for choosing HUTN for ASEME.

A T2M transformation is used for transforming a liveness formula to a statechart (IAC model). We first use an iterative algorithm (see [26]) that creates the HUTN model, which is then automatically transformed to an IAC model. The usage of the HUTN technology also helped a lot in debugging the algorithm as the output was in human-readable format.

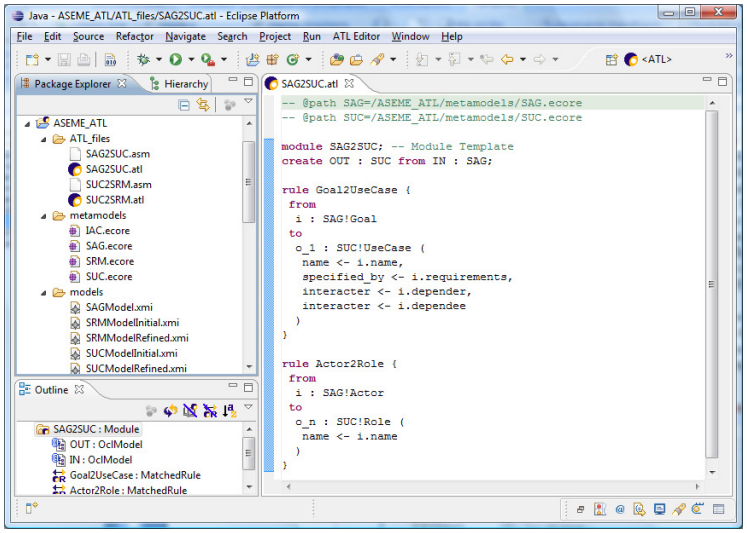


Fig. 4. The eclipse ATL project for the SAG2SUC and the SUC2SRM M2M transformations.

ASEME M2T Transformation Tool (IAC2JADE)

The last transformation type used in the ASEME process is M2T. The platform independent IAC model must be transformed to a platform dependent one and to executable code. We used the Xpand language offered by the Eclipse. Another commonly used M2T transformation language (in EMP) is the Java Emitter Templates (JET). JET uses JSP-like templates, thus it is easy to learn for developers familiar with this technology.

The advantages of Xpand are the fact that it is source model independent, which means that any of the EMP parsers can be used for common software models such as MOF or EMF. Its vocabulary is limited allowing for a quick learning curve while the integration with Xtend allows for handling complex requirements. Then, EMP allows for defining workflows that allow the modeler to parse the model multiple times, possibly with different goals.

In ASEME, the developer uses the IAC2JADE tool that automatically generates the message receiving and sending behaviours and the composite behaviours that coordinate the execution of simple behaviours. Thus, the user just needs to program the action methods of simple behaviours.

5 Related Work and Conclusion

A number of works in AOSE have introduced concepts and ideas from the model-driven engineering domain. Most of them just introduce an MDE technique for transforming one of their models to another in one phase, e.g. from a Tropos plan decomposition diagram to a UML activity diagram in [18] and from a BDI (Belief-Desire-Intention) representation in XML format to JACK platform code in [9]. Almost all AOSE methodologies define a single, usually huge metamodel covering all the requirements, analysis and design phases [1].

Other works aim to create a single meta-model that can be used by different AOSE methodologies in a specific phase, like in [6], where the authors defined a meta-model (PIM4Agents) that can be used to model MAS in the PIM level of MDA, and in [1], where the authors try to envisage a unifying MAS metamodel. Finally, a more recent work [4] presents an algorithm to generate model transformations by-example that allows the engineer to define himself the transformations that he wants to apply to models complying with the INGENIAS metamodel.

ASEME furthers the state of the art by being the first AOSE methodology to propose a model-driven approach covering all the development phases and incorporating three types of transformations (M2M, T2M, M2T). Moreover, in ASEME the information to be added at each phase is clear and the models used are common in the software engineering community, which means that any engineer can quickly adapt to the ASEME process. Model transformations are automated throughout the software development process. In the previous sections, we presented the formal definition of the AMOLA metamodels, which have been inspired by previous works but are original in the way that they uniquely extend those works and insert new semantics, thus assisting the ASEME process. We also presented the models transformations that occur in the different phases of ASEME. The platform independent model of ASEME, i.e. the IAC, is a statechart which can be transformed to a platform specific model in C++ or Java (using commercial CASE tools) or in the JADE agent platform. This is another originality of ASEME, it is the first AOSE methodology to provide a PIM model that is compatible with existing software tools (i.e. the statechart) giving multiple platform choices to the developers.

ASEME has been successfully used for the development of two real world systems ([13], [25]). Table 1 shows a quick comparison of ASEME with existing AOSE methodologies. It has been inspired by a similar table in [28] from which we use some criteria (rows). The first row shows the levels of abstraction supported by the methodologies. Only ASEME maintains three levels of abstraction throughout the software development phases. Some do not support abstraction at all, while others do a phase-based abstraction (e.g. define agent interactions and roles in the analysis phase and focus in the specific agent development in the design phase). The next row shows the MDE support for the different software development phases. ASEME supports all the phases, many methodologies support some phases and INGENIAS allows the modeler to define his own transformations. The third row shows if a methodology covers all the software development phases, i.e. requirements analysis, system analysis, design, implementation, verification and optimization. The forth row

shows what kind of agents each methodology supports and the fifth row indicates which methodologies define an intra-agent control model that allows an agent to coordinate his capabilities, thus supporting a modular development approach. Finally, the sixth row shows that ASEME is the only methodology to use a uniform representation of inter-agent protocols and the intra-agent control allowing for an easy integration of protocols in an agent specification. In Table 1 "n/a" means not applicable.

Table 1. ASEME compared with existing AOSE methodologies.

Methodology	ASEME	Gaia	Tropos	INGENIAS	PASSI	Prometheus	MaSE
Abstraction	all phases	n/a	phase-based	n/a	phase-based	phase-based	n/a
MDE phases	all	n/a	some	defined by the modeler	some	n/a	some
Phases coverage	all	some	all	some	some	some	some
Agent nature	hetero-geneous	hetero-geneous	BDI-like agents	agents with goals and states	hetero-geneous	BDI -like agents	not specified
Intra-agent control (IAC)	yes	no	no	no	no	no	yes
Uniform representation of IAC and inter-agent protocols	yes	no	no	no	no	no	no

References

1. Bernon, C., Cossentino, M., Pavon, J.: Agent Oriented Software Engineering. The Knowl. Eng. Rev., 20, 99--116 (2005)
2. Budinsky, F., Steinberg, D., Ellersick, R., Merks, E., Brodsky, S.A., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley (2003)
3. Efftinge, Sven and Völter, Markus. oAW xText: A framework for textual DSLs. In Eclipse Summit 2006 Workshop: Modeling Symposium (2006)
4. García-Magariño, I., Gómez-Sanz, J.J., Fuentes-Fernández, R.: Model Transformations for Improving Multi-agent Systems Development in INGENIAS. In: 10th International Workshop on Agent-Oriented Software Engineering (AOSE'09), Budapest Hungary (2009)
5. Gerber, A., Raymond, K.: MOF to EMF: there and back again. In: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange (eclipse '03), pp. 60--64. ACM Press, New York (2003)
6. Hahn, C., Madrigal-Mora, C., and Fischer, K.: A platform-independent metamodel for multiagent systems. J. Auton. Agents and Multi-Agent Syst., 18, 2, 239--266 (2009)
7. Harel, D., Kugler, H.: The RHAPSODY Semantics of Statecharts (Or on the Executable Core of the UML). In: Integration of Software Specification Techniques for Applications in Engineering. LNCS, vol. 3147, pp. 325--354. Springer, Heidelberg (2004)
8. Henderson-Sellers B. and Giorgini P.: Agent-Oriented Methodologies. Idea Group Publishing (2005)
9. Jayatilleke, G.B., Padgham, L., and Winikoff, M.: A model driven component-based development framework for agents. Int. J. Comput. Syst. Sci. Eng. 20, 4, 273--282 (2005)
10. Jouault, F., Bézivin, J.: KM3: A DSL for Metamodel Specification. In: Formal Methods for Open Object-Based Distributed Systems (FMOODS 2006). LNCS, vol. 4037, pp. 171--185. Springer, Heidelberg (2006)

11.Jouault, F. and Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. LNCS, vol. 3844, Springer-Verlag, pp. 128--138 (2006)
12.Langlois, B., Jitia, C.E., Jouenne, E.. DSL Classification. In 7th OOPSLA Workshop on Domain-Specific Modeling, URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.133.136 (2007)
13.Moraitis, P., Spanoudakis N.. Argumentation-based Agent Interaction in an Ambient Intelligence Context. IEEE Intell. Syst. 22, 6, 84--93 (2007)
14.Object Management Group: Meta Object Facility (MOF) Core Specification (2001)
15.Object Management Group: Human-Usable Textual Notation V1.0 (2004)
16.Object Management Group: Revised Submission for MOF 2.0 Query/View/Transformations RFP, OMG Document ad/2005-07-01 (2005)
17.Object Management Group: Software & Systems Process Engineering Meta-Model Specification, version 2.0 (2008)
18.Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented Modeling. In: Agent-Oriented Software Engineering VI. LNCS, vol. 3950, pp. 167--178. Springer (2006)
19.Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.: Constructing models with the Human-Usable Textual Notation. In: 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS). LNCS, vol. 5301, pp. 249--263. Springer (2008)
20.Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Softw. 20(5), 42--45 (2003)
21.Spanoudakis, N.: The Agent Systems Engineering Methodology (ASEME). Philosophy Dissertation. Paris Descartes University, Paris, France, URL: http://users.isc.tuc.gr/~nispanoudakis/SpanoudakisThesis.pdf (2009)
22.Spanoudakis N., Moraitis, P.: The Agent Systems Methodology (ASEME): A Preliminary Report. In: Proc. of the 5th European Workshop on Multi-Agent Systems (EUMAS'07), Hammamet, Tunisia, December 13 - 14 (2007)
23.Spanoudakis, N. and Moraitis, P.: The Agent Modeling Language (AMOLA). In: Proceedings of the 13th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA 2008), LNCS, vol. 5253, pp. 32--44. Springer (2008)
24.Spanoudakis, N. and Moraitis, P.: An Agent Modeling Language Implementing Protocols through Capabilities. In: Proceedings of The 2008 IEEE/WIC/ACM Int. Conference on Intelligent Agent Technology (IAT-08) , Sydney, Australia, December 9-12, (2008)
25.Spanoudakis N., Moraitis, P.: Automated Product Pricing Using Argumentation. In: Iliadis, L.; Vlahavas, I.; Bramer, M. (Eds.), IFIP Advances in Information and Communication Technology Book series, Vol. 296, Springer (2009)
26.Spanoudakis, N., Moraitis, P.: Gaia Agents Implementation through Models Transformation. In: Proceedings of the 12th International Conference on Principles of Practice in Multi-Agent Systems (PRIMA 2009), LNAI, vol. 5925, pp. 127--142 (2009)
27.Susi, A., Perini, A., Giorgini, P. and Mylopoulos, J.. The Tropos Metamodel and its Use. Inform. 29, 4, 401--408 (2005)
28.Tran, Q.N.N., Low, G.C.: Comparison of ten agent-oriented methodologies. In [8] (2005)
29.Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: the Gaia Methodology. ACM T. Softw. Eng. Meth. 12(3), 317--370 (2003)

Towards the automatic derivation of Malaca agents using MDE*

Inmaculada Ayala, Mercedes Amor and Lidia Fuentes

E.T.S.I. Informática, Universidad de Málaga
{ayala,pinilla,lff}@lcc.uma.es

Abstract. The automatic transformation of software agent designs into implementations for different agent platforms is currently a key issue in the MAS development process. Recently several approaches have been proposed using model driven development concepts to specify generic agent metamodels and/or define a set of transformation rules from the design phase for different agent implementation platforms. However, all these approaches propose different sets of transformation rules for each target agent platform, thereby making the integration of new agent platforms more difficult. In this paper we propose to transform PIM4Agents, a generic agent metamodel used at the design phase, into Malaca, an agent specific platform-neutral metamodel for agents. With only one set of transformations it is possible to specify platform-neutral agents and to generate a partial implementation in Malaca, which can be executed on top of different FIPA compliant platforms.

Keywords: Agent Oriented Software Engineering, Model driven engineering, Malaca, PIM4Agents, Code generation

1 Introduction

In order to make agent-based computing a widely accepted paradigm for the emerging application areas, advanced development processes of software engineering should be adopted. This process must be supported by agent development tools that alleviate the complexity of programming with agent platforms (APs) by providing facilities to express domain concepts at a higher level of abstraction.

Model-Driven Engineering (MDE) [1] is an approach for Software Development that promotes the use of models and metamodels to formally represent domain concepts. One important contribution of MDE is that a software system is obtained through the transformation of different metamodels defined at different abstraction layers. These transformations, defined by means of a model transformation language, allow deriving a PSM (Platform-Specific Metamodel) from a PIM (Platform-Independent Metamodel). MDE ideas can bring important benefits to the development

* This work has been supported by the Spanish Ministry Project RAP TIN2008-01942 and the regional project FamWare P09-TIC-5231.

of Multi-Agent Systems (MAS) as shown in [2, 3, 8,16]. With MDE it is possible to specify a MAS in a platform-independent model, focusing on the domain model, and later transform it automatically to different design or implementation models, bridging the traditional gap between design and implementation.

One recent and notable effort in this direction is [4]. This work proposes a PIM for MAS (PIM4Agents) and a set of vertical transformations from this metamodel to different AP models, concretely JADE and JACK. However, this work has some drawbacks that we will address in this paper. Although PIM4Agents can be used in theory to derive PSMs for any AP, in practice the DSL4MAS Development Environment (DDE) tool [10] provided by the authors only supports the transformation to JADE and JACK. This means that other APs which have emerged recently such as Andromeda [5], or different versions of JADE (e.g. LEAP, Android) are not currently covered by this proposal. But, what is the cost of including a new AP in this proposal? It requires the definition of a new set of transformation rules, from PIM4Agents into the metamodel of the new AP, and from the new AP metamodel into code. This is a very complex task, sometimes impossible to perform properly in this and in other approaches [6] due to: (i) the metamodel of the target AP must be available, which is not always the case; (ii) sometimes the target metamodel is not specified completely, and some mappings to the target metamodel are made in an ad-hoc manner; (iii) this task also requires some expertise in a transformation language; and (iv) also the transformations from the target AP metamodel to code have to be implemented, requiring an depth knowledge of the target agent implementation framework.

In order to bridge the gap from design to implementation, we previously defined a set of transformations from different agent methodologies to Malaca, an agent model able to be executed in different APs [13]. Although Malaca can be used as an agent model at the detailed design phase, its model is also a PSM. Nevertheless, we had a similar problem to [4], since we had to define different transformation rules to go from different design models (e.g. Tropos) to Malaca. As a solution in [8] we proposed to define a generic agent metamodel modelling the most commons elements covered by the existing agent-methodologies. In this direction, and instead of defining a new MAS metamodel, we studied the feasibility of using one of the works proposed recently [4, 9]. Finally we decided to use PIM4Agents since this metamodel meets the following requirements: (i) it is possible to represent concepts from different agent types (e.g. BDI, reactive agents), (ii) it is easy to specify MAS for different domains; (ii) the DDE tool helps to specify different views of MAS.

Therefore, as a solution for the automatic derivation of MAS to different APs, in this paper we propose to transform specifications from PIM4Agents to Malaca. Malaca applies aspect-oriented principles (AOSD[†]) to separate the distribution of messages according to different transport services in a *distribution aspect* (implemented as a plug-in), which reduces AP dependency inside the agent architecture.

[†] Aspect-Oriented Software Development (AOSD) http://aosd.net/

The structure of this paper is as follows. Section 2 provides a brief overview of MDE, and it introduces PIM4Agents and Malaca metamodels. Section 3 describes our main contribution, by showing the transformation rules implemented in ATL to transform agents from the PIM4Agents metamodel to Malaca and we illustrate how to use them with an example. Section 4 outlines some of the problems and limitations of our approach. Finally, Section 5 provides related work and Section 6 draws some conclusions.

2 Background

In this section we introduce the concept of MDE and the two agents' metamodels used by our approach, the PIM4Agents and Malaca metamodels (MalacaMM).

2.1 Model Driven Engineering

MDE is an approach for Software Development where models are now first class citizens of the software development process, and even the code is managed as a model. The best known MDE initiative is the OMG initiative Model-Driven Architecture (MDA). In MDA, PSMs are derived from PIMs of the upper abstraction layer, by means of *model transformations* (i.e. how an output model is constructed based on the elements of an input model).

An important advantage of the application of MDA is that model transformations expressed in a well-defined model transformation language can be compiled and executed, automating the process of constructing a target model for a given source model. ATL (ATLAS Transformation Language) [7] is a model transformation language. ATL provides ways to produce a set of target models from a set of source models. Developed on top of the Eclipse platform, the ATL Integrated Environment provides a number of standard development tools that aims to ease development of ATL transformations. It also includes a library of ATL transformations.

2.2 PIM4Agents

Domain Specific Modelling Language for MultiAgent System (DSML4MAS) [10] is an approach that tries to fill the gap between agent methodologies and agent-based development tools by using MDE principles. This approach provides PIM4Agents, which is a PIM for MAS, and a tool (DDE) that provides a graphical modelling framework to design MAS. The PIM4Agents metamodel tries to include concepts that are present in most agent architectures. It has several views: Multiagent view, Agent view, Behavioural view, Organisation view, Role view, Interaction view and Environment view. Since we are interested in deriving agent designs we will focus on the Agent and the Interaction views. Fig. 1 shows the metamodel used by the agent view.

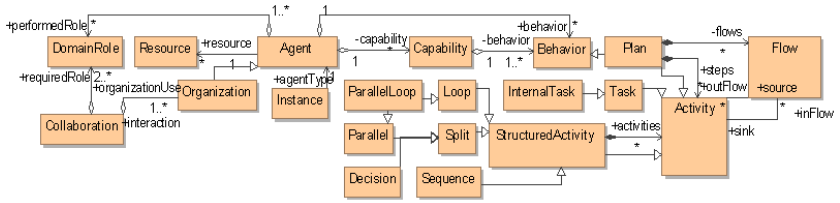


Fig. 1: Metamodel reflecting agent aspect

In PIM4Agents, an agent is an autonomous entity capable of acting in the environment, which can access a set of resources and perform some domain roles derived from its collaboration with other agents. Agents can also perform a set of *Behaviors*, which can be separated into a set of internal processes represented by *Plan* elements. A *Plan* is composed by a set of *Flow* and *Activity*. *Activity* elements can be simple (*Task*) or complex (*StructuredActivity*). In the interaction view metamodel (Fig. 2) the main component is the *Protocol*, which represents the interaction between a set of *Actor*. An *Actor* has a set of *activeStates*, which corresponds to *MessageFlow*, and specifies how the exchange of messages is performed. A *MessageScope* defines the *Messages* and the order in which these arrive.

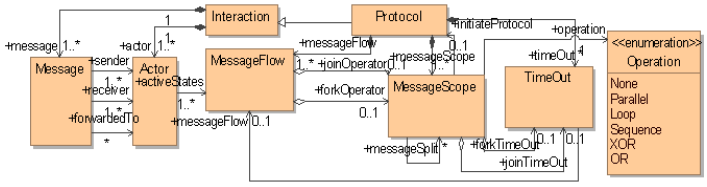


Fig. 2: Partial metamodel reflecting interaction aspect

2.3 Malaca

Most existing agent architectures focus on the type of agent (BDI, reactive), but do not provide direct support for handling and reusing properties and functionality separately. This approach results in agent design and implementations being quite complex, brittle and difficult to understand, maintain, and reuse in practice. The main feature of the internal architecture of a Malaca [13] agent is that it represents separately application-specific functions from extra-functional agent properties. This separation improves the internal modularization of the agent architecture, which is based on the composition of components and aspects, and contributes to enhance the adaptation, reuse and maintenance of the software agent. The Malaca agent model is used from the detailed design phase right through to implementation. At the detailed design stage two XML-based domain-specific languages (MaDL and ProtDL) are used to design the internal architecture of each agent of the system and its interaction [13]. MalacaMM, which is partially given in Fig. 3, presents the concepts and constructs available in MaDL to describe an agent architecture.

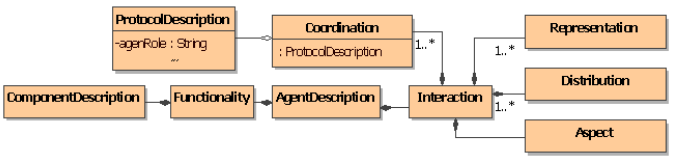


Fig. 3. (Partial view of) the Malaca Metamodel.

The *AgentDescription* provides a description of the agent architecture in view of the agent functionality (the actions that an agent is able to perform described by means of components) and the agent interaction (how the agent communicates with other agents, which is provided by different aspects). The agent includes reusable software components, which offer the set of core services, and also the application-dependent functionality.

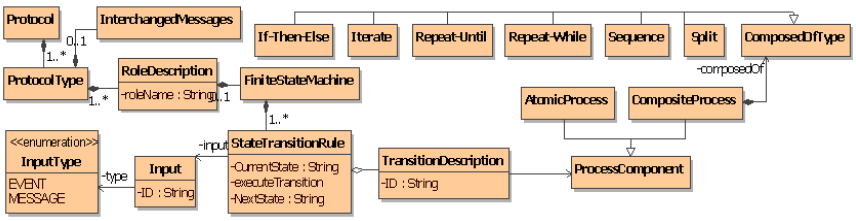


Fig. 4. (Partial view of) the protocol and process metamodel.

The way the agent interacts is described at the architectural level by a set of aspects. Each aspect covers a different property of agent communication. Based on the FIPA communication model, three issues are considered essential for an effective communication: the use of an interaction protocol, a common language representation format for the ACL and a MTS (Message Transport Service) to distribute messages. In Malaca, each one is supplied by different aspects (coordination, representation and distribution) decoupling these interaction issues. Specifically, the distribution aspect copes with the use of MTS, facilitating the use of different AP just by plugging in the agent architecture the aspect implementing a specific AP. Any interaction protocol supported by the agent is controlled by a coordination aspect (class *Coordination*). This aspect uses a description of the interaction protocol in ProtDL to coordinate message interchange with the agent internal behaviour. The description of this aspect also indicates the role played by the agent. The UML class diagram in Fig. 4 depicts the metamodel of a protocol description in ProtDL (ProtDLMM), which includes a description of the ACL messages interchanged during the interaction and a description of the internal behavior of each participant role (*RoleDescription* element). A finite state machine (FSM) is used to represent each participant role. Each FSM is represented by a set of state transition rules enclosed by the *FiniteStateMachine* class and each rule is defined in a *StateTransitionRule* class. The transition from a state to another carries out the execution of the agent functionality

(defined in the *StateTransitionRule* by the attribute *executeTransition*). The *TransitionDescription* class encloses the description of the set of agent actions that are invoked during protocol execution using a *Process* model (Fig. 4). A *TransitionDescription* carries out the description a *ProcessComponent*, which can be either a single (or atomic) action, or a composite process composed of a set of processes related by typical control construct.

3 From PIM4Agents to Malaca

MDE ideas and techniques enhance AOSE enabling reuse at the domain level. The DSML4MAS approach applies MDE and using PIM4Agents as a PIM provides a set of mapping functions to transform PIM4Agents model to JACK and JADE (see Fig. 5). However, one of the problems in this approach is found when trying to implement the MAS for an AP different from these. This decision requires expert knowledge to derive the appropriate mappings to the new AP. To solve this problem, we propose a mapping from PIM4Agents to Malaca, an agent architecture that can be executed on top of any AP using the appropriate plug-in. An overview of our approach can be seen in Fig. 5 (right side).

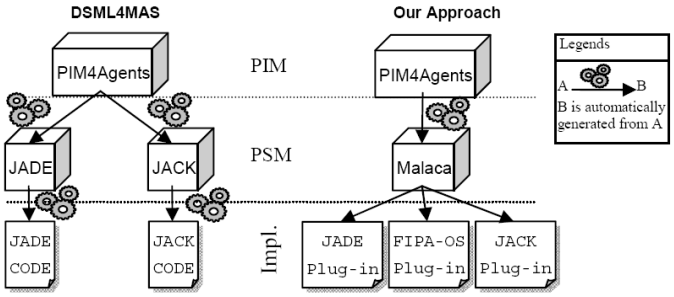


Fig. 5: The overall picture: From PIM4Agents metamodel to Malaca metamodel.

With this approach we obtain the benefits of using a general platform independent metamodel to specify the design of a MAS, and transform it (using the set of transformation rules presented in this paper) automatically into a set of Malaca agents in accordance with MalacaMM. The benefit of using Malaca as PSM is twofold: The incorporation of new APs (i.e. PSMs) to this proposal has a lower cost since instead of requiring the specification of PSM metamodels and coding a new set of transformations rules, only the implementation of a new plug-in is needed; and the implementation of a MAS for different APs does not require transforming and implementing it for each AP, instead, it just involves selecting and using the appropriate AP plug-in for each Malaca agent.

3.1 Transformation rules

In the MalacaMM two main parts can be distinguished: the specification of the agent architecture (Fig. 3); and the specification of interaction protocols (Fig. 4), in MaDL and ProtDL languages respectively. Table 1 summarizes main mappings between PIM4Agents concepts and MaDL concepts. In the following section we will focus on the transformation of coordination-related concepts in PIM4Agents to ProtDL protocols descriptions. This section introduces this transformation, which requires several ATL mapping rules.

Table 1. Process mapping between the PIM4Agents concepts and MaDL concepts

Target	Source	Explanation
AgentDescription	Agent	Each agent in PIM4Agents is mapped to an Agent in MaDL.
Functionality	InternalTask	Each InternalTask from a Plan associated to an Agent in PIM4Agents is a Functionality in MaDL.
Coordination	Protocol	Each Protocol associated to an Agent means of a Collaboration is mapped to a Coordination.
Distribution		JADE-mts by default.
Representation		FIPA-ACL by default.

The mapping rules included do not constitute an exhaustive list. We have only included those that help to comprehend the most relevant model mappings required for the use case scenario. Each mapping rule consists of (i) a head that defines which concepts from the source metamodel are mapped to which concepts of the target metamodel and (ii) a body that defines how attribute information of the target metamodel is derived. Some mapping rules are applied automatically (simple ATL rules), while the application of other rules depends on the previous application of other mapping rules or must be invoked by other rules (ATL lazy rules).

MR 1: Head: *PIM4Agents!Protocol* → *ProtDLMM!Protocol*
Body: Each Protocol from PIM4Agents is mapped to a Protocol in Malaca.

This rule maps a *PIM4Agents!Protocol* to a *Protocol* in Malaca with the same *ID*, interchanged messages and actors.

MR2: Head: *PIM4Agents!Actor* → *ProtDLMM!RoleDescription*
Body: Each *Actor* is mapped to a *RoleDescription* associated to a specific *Protocol*

Each *Actor* in the *PIM4Agents!Protocol* is mapped to a *RoleDescription* in *ProtDLMM!Protocol*. The *RoleDescription* has the same *ID* as the *Actor*. The *Actor activeStates* (Fig. 2) are mapped to the states of the *RoleDescription's FiniteStateMachine*.

MR3: Head: *PIM4Agents!MessageFlow, PIM4Agents!MessageFlow* → *ProtDLMM!StateTransitionRule*

Body: From two *MessageFlow* this rule creates a *StateTransitionRule* that begins in the first *MessageFlow* and ends in the second one.

This rule is linked to *MR2* and it is used to derive the *StateTransitionRule* of a *FiniteStateMachine*. The occurrence of two consecutive *MessageFlows* (for a given *Protocol* and *Actor*) is mapped to a *StateTransitionRule*. The first *MessageFlow* is mapped to the current state while the second one is mapped to the next state.

MR4: Head: *PIM4Agents!MessageFlow, PIM4Agents!MessageFlow* → *ProtDLMM!TransitionDescription*

Body: From two *MessageFlow* this rule creates a *TransitionDescription* that begins in the first *MessageFlow* and ends in the second one.

This rule is linked to *MR3* and it is very similar to *MR3*. The occurrence of two consecutive *MessageFlows* is mapped to a *TransitionDescription* which describes a message sending.

MR5: Head: *PIM4Agents!Plan,String* → *ProtDLMM!RoleDescription*

Body: Creates a *RoleDescription* from a *Plan* and a *String* that is the name for the *Role*.

This lazy rule maps a *Plan* (associated to a given *Actor* or *Agent* denoted by the *String* that is passed as an argument) to *RoleDescription*, identified with the same *String*. During its application, *MR8* needs a special function (helper) to ignore *ReceiveMessage Activity* (Malaca does not consider it as a *Process* but as a *MESSAGE InputType* for a *StateTranstionRule*). Then the *MR9* is applied.

MR6: Head: *PIM4Agents!Activity, PIM4Agents!Activity* → *ProtDLMM!StateTransitionRule*

Body: From two *Activity* this rule creates a *StateTransitionRule* that begins in the first *MessageFlow* and ends in the second one.

This rule is very similar to *MR4* but it considers *Activities* instead of *MessageFlows* to generate *StateTransitionRule*(s) of the *FiniteStateMachine*.

MR7: Head: *PIM4Agents!Activity, PIM4Agents!Activity* → *ProtDLMM! TransitionDescription*

Body: From two *Activity* this rule creates a *TransitionDescription* that begins in the first *Activity* and ends in the second one.

This rule maps an *Activity* (or a *StructuredActivity*) to a *TransitionDescription*. The description of the *TransitionDescription* is derived from the application of the next rules.

MR8: Head: *PIM4Agents!InternalTask* → *ProtDLMM!ProcessComponent*

Body: Each *InternalTask* is mapped to a *ProccesComponent* that have an *AtomicProcess* whose type is *DoActionType*.

The rule maps a PIM4Agents *InternalTask* of to a Malaca *DoAction* atomic process.

MR9: Head: *PIM4Agents!Split* → *ProtDLMM!ProcessComponent*
Body: Each *Split* is mapped to a *ProccesComponent* that have a *CompositeProcess* whose type is *SplitType*.

Each PIM4Agents *StructuredActivity* is mapped to a ProtDLMM *CompositeProcess*. As an example, this rule maps *Split* to a *ProccessComponent* with a *CompositeProcess* that is a *Split*. *MR8* is used to map the *BasicTasks* of the *StructuredActivity*.

MR10: Head: *PIM4Agents!Protocol, PIM4Agents!Organization* → *ProtDLMM!Protocol*
Body: Each *Protocol* which is from an *Organization* is mapped to a *Protocol*.

This rule maps each PIM4Agents *Protocol* within an *Organization* to a ProtDLMM *Protocol*. The application of this rule generates a *Protocol* which includes a *RoleDescription* which corresponds to a set of actions describing the behaviour of the agent during the interaction. If there is no PIM4Agents *Plan* associated to the protocol, then just the message interchanged is described (*MR3*).

3.2 Use case scenario

To illustrate the MDE process, the Conference Management System (CMS) case study will be used. This case study was used and evaluated in our previous work [14] and it was also used in the PIM4Agents work [4]. The design of the CMS system has been derived from the DDE tool using a tutorial [15]. The full example consists of 7 diagrams but for simplicity we will only consider the diagrams shown in Fig. 6 and Fig. 7, which corresponds to a protocol and a plan.

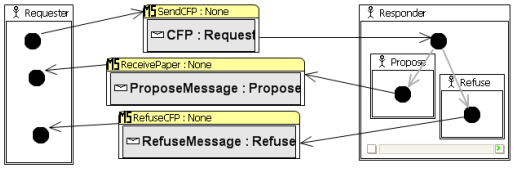


Fig. 6. PIM4Agents Protocol Diagram of the PaperSubmission protocol.

Fig. 6 shows the protocol diagram of the *PaperSubmission* protocol, which covers the interaction between the *Requester* and *Responder* actors in the submission phase. *Requester* sends a *CFP* that can be answered by *Responders* with a *Propose* or a *Refuse* message. Fig. 7 depicts the plan *HandleCFP*, which is executed by the *Responder* when it receives a *CFP*, and it decides whether to submit a paper or to send a refuse message and relax. Fig. 8 presents a partial result of the application of rules *MR10*, *MR5*, *MR6* and *MR7* (in this order) to the *Responder* actor in the *PaperSubmission* protocol. After the application of the rule *MR10* to the protocol diagram of Fig. 6 (and related organization diagram), the rule *MR5* is applied to the *HandleCFP* plan (Fig. 7) and generates a ProtDL *RoleDescription* for the role

Responder. Rules *MR6* and *MR7* are also applied to the *HandleCFP* plan to derive the states and *StateTransitionRules* of the *FiniteStateMachine* of the *Responder*.

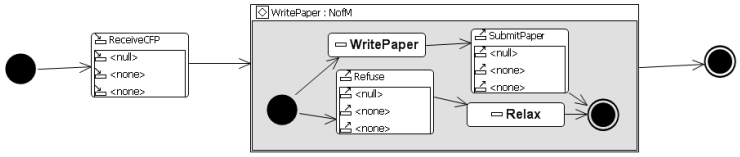


Fig. 7. PIM4Agents Plan diagram for the Handle CFP plan.

```
<roleDescription roleName="Responder">
 <finiteStateMachine>
  <stateDeclaration>
   <state>Begin</state>
   <state>End</state>
   <state>WritePaper</state>
   <finalState>End</finalState>
  </stateDeclaration>
  <stateTransitionRule currentState="Begin" nextState="WritePaper" name="STRule Begin to WritePaper">
   <input value="value" ID="Message to WritePaper" type="MESSAGE"/>
   <executeTransition value="value" ID="Begin execution "/>
  </stateTransitionRule>
  <stateTransitionRule currentState="WritePaper" nextState="End" name="STRule WritePaper to End">
   <input value="value" ID="WritePaper end" type="EVENT"/>
   <executeTransition value="value" ID="WritePaper execution "/>
  </stateTransitionRule>
```


Fig. 8. ProtDL Role description of the Requester role (in Paper Submission protocol).

4 Discussion

This section shows some comparative results between the code generated with DDE tool and the configuration files generated by our approach, for the CMS case study. Although we have defined a systematic mapping from PIM4Agents to Malaca, it is not possible to generate a complete ProtDL specification since: (i) DDE does not support the specification of significant message details (at least a rough description) and protocol design features; and (ii) some typical fields of protocol specification (such as the ontology used by the ACLMessage) are not supported either by the PIM4Agents metamodel or by the tool (they are supported by different design metamodels). In addition, some concepts related to the MAS social organization cannot be mapped to Malaca (such as the Organization concept), because Malaca is focused on the specification and configuration of the agents internal architecture. Hence, some abstractions of PIM4Agents could not be mapped in Malaca; and some concepts present in Malaca could not be generated from the PIM4Agents metamodel. Then, the generated MaDL and ProtDL descriptions have to be completed by the developer (using the Malaca editor [13]) before executing the agents.

This also happens in the implementations generated by the DDE tool (applying transformations to JADE and JACK as target implementation APs). For the case

study, which comprises the design of one protocol and two plans, the DDE tool generates 22 Java classes (for the JADE implementation). The generated classes provide the agent structure, and the developer has to complete the implementation classes. Comparing both approaches, we consider that completing the Java classes generated with DDE tool is more error prone than using the Malaca Agent Development (MAD) tool to complete the configuration of the Malaca agents. Whereas in DDE approach the developer has to deal with 22 classes, in the Malaca approach only has to complete a single configuration file with the MaDL/ProtDL specification, with the aid of the MAD tool.

On the other hand, the code generated with DDE is not optimized. For example, in JADE several template classes encapsulating FIPA protocols are provided, but the DDE generator does not consider these templates in the code generation. Malaca offers similar template files for FIPA protocols, and the configuration files generated are optimized using these templates. As a result, it is possible to reuse protocol specifications in several case studies, avoiding the generation of code for the same protocol once again. Also, it is possible to reuse some protocol information, such as the ontology used, or the message content, for similar case studies, but in DDE this is not possible. Hence, completing protocol implementation classes, scattered across several classes, requires greater effort in the DDE approach.

5 Related Work

There are some approaches that apply MDE concepts to AOSE in different contexts. The Gaia methodology [6] defines a specific mapping to JADE as PSM, but it is not an automatic process. Different agent oriented methodologies, such as MaSE[17] support a complete tool-aided life cycle process from early requirements to code generation. Moreover, in some of them, such as Tropos[16] and INGENIAS[3], the life-cycle is an MDE process. MDE is also approached in [5], which applies MDE for mobile agents. It takes Agent-π, a metamodel for mobile devices, as PIM and provides transformations to two mobile-specific PSMs, Andromeda and JADE-Leap. Although the intention of these approaches was to cover the implementation phase, they have the same disadvantages as those mentioned for the PIM4Agents approach: (i) a different transformation is needed for every PSM; and (ii) the implementation of agents in JADE and other OO agent architectures is difficult to maintain and reuse. The problem is that normally the agent internal architecture consists of a collection of highly-coupled objects, making it difficult to extend. Since different agent concerns such as the agent domain-specific functionality are not very well modularized, each time the agent needs to be upgraded, the developer must inspect the implementation code, then change and re-compile it. An additional disadvantage of these approaches is that they use their own agent metamodel (and not a generic one) in the design phase. Recently, a proposal was made to define a generic agent metamodel, FAML[18]. Unlike PIM4Agents model, which was defined from an AP, FAML is a unified metamodel defined considering the concepts present in the metamodel of several agent methodologies. However, it has no tool support yet.

6 Conclusion

This paper presents an MDE approach to developing MAS using the PIM4Agents metamodel as PIM and Malaca as PSM, focusing on the external and internal coordination of agents. Following an MDE approach we have defined mapping rules to generate a set of MaDL/ProtDL files. From these, implementation details are added and can be used to deploy and execute Malaca agents. The derivation of agents for new APs is enhanced in this approach, since it is accomplished by simply selecting the appropriate distribution aspects. In other approaches, this has to be done by defining new agent platform metamodels, and the corresponding transformation rules. We are currently integrating the mapping rules presented in this paper in MAD tool.

7 References

1. A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture–Practice and Promise*. Addison-Wesley Professional, April 2003.
2. F. Zambonelli, and A. Omicini, *Challenges and Research Directions in Agent-Oriented Software Engineering*, Auton Agent Multi-Agent Syst (2004), 9:253–283.
3. J. Pavón, J. Gómez-Sanz and R. Fuentes, *Model Driven Development of Multi-Agent Systems*, Proceedings of the ECMDA-FA conference, LNCS, 4066/2006
4. C. Hahn, C. Madrigal-Mora, and K. Fischer. *A platform-independent metamodel for multiagent systems*, Auton Agent Multi-Agent Syst (2009) 18:239–266
5. Agüero, J., Rebollo, M., Carrascosa, C., and Julián, V. (2009). *Agent Design Using Model Driven Development*. PAAMS'09, AISC 55, pp. 60–69.
6. Moraitis, P., & Spanoudakis, N. I. (2006). The Gaia2Jade process for multi-agent systems development. Applied Artificial Intellige1nce, 20(2–4), 251–273.
7. ATL. http://www.eclipse.org/m2m/atl/
8. Amor M., Fuentes L. and Vallecillo A. *Bridging the gap Between Agent–Oriented Design and Implementation Using MDA*, In AOSE 2004, LNCS 3382, pp. 93–108, 2005.
9. Beydoun, G., et al. *Synthesis of a generic mas metamodel*. In SELMAS 05. pp. 1–5.
10. Stefan Warwas, Christian Hahn. *The DSML4MAS Development Environment*, in Proc. AAMAS 2009, pp. 1379-1380.
11. Paolo Busetta, et al, *JACK Intelligent Agents - Components for IntelligentAgents in Java*. Tech. Rep. Agent Oriented Software 1998.
12. Bellifemine, F., Rimassa, G., Poggi, A., *JADE - A FIPA-compliant Agent Framework*. In Proc. of PAAM 1999.
13. M. Amor and L. Fuentes. *Malaca: A component and aspect-oriented agent architecture*. In Information and Software Technology 51 (2009) 1052–1065.
14. M. Amor, L. Fuentes, J. Valenzuela, *Separating learning as an aspect in Malaca agent*s, in: KES-AMSTA, LNCS 4953, 2008, pp.505-515.
15. DDE tool. http://sourceforge.net/apps/trac/dsml4mas/wiki
16. Susi, A., Perini, A., Mylopoulos, J. *The Tropos Metamodel and its Use*. Informatica 29, 401–408 (2005)
17. S. A. DeLoach and M. Wood, *Developing Multiagent Systems with agentTool*, in ATAL 2000. LNAI , 2001.
18. G. Beydoun, et al., *FAML: A Generic Metamodel for MAS Development*, IEEE Transactions on Software Engineering, 99, pp. 841-863, 2009

$\mathcal{F}_{or}\mathcal{MAAD}$: Towards A Model Driven Approach For Agent Based Application Design

Zeineb Graja, Amira Regayeg, and Ahmed Hadj Kacem

ReDCAD Laboratory
Faculty of Economics and Management
University of Sfax, Tunisia
zeineb.graja@acm.org
{amira.regayeg,ahmed.hadjkacem}@fsegs.rnu.tn

Abstract. Current trends in multi-agent systems development show a move towards adopting the Model Driven Architecture (MDA) approach to improve the development process and the quality of the agent-based software. Our work has two main contributions. First, it presents a reformulation of the $\mathcal{F}_{or}\mathcal{MAAD}$ methodology in terms of the MDA paradigm by using the AML language. Second, it proposes a transition of each model to a formal language, $\mathcal{T}_{emporal}\mathcal{Z}$ that integrates linear temporal logic to the Z notation, in order to guarantee a formal verification of the models. Furthermore, we make extensions to the *StarUML* tool to support the proposed models and use the transition rules. Our work is illustrated by developing an agent-based solution for the air traffic control problem.

Key words: MDA, AML language, formal methods, refinement,verification

1 Introduction

Current trends in Multi-Agent Systems (MAS) development show a move towards adopting the MDA approach to improve the development process and the quality of the agent-based software ([1], [2], [3], [4]). The basic motivation of MDA is that it allows improvement of an application development process. In fact, MDA suggests to use model transformation techniques to generate automatically PSM (Platform Specific Model) from PIM (Platform Independent Model).

The work presented in this paper consists of a reformulation of $\mathcal{F}_{or}\mathcal{MAAD}$ methodology, based on a formal framework and dedicated for the design of multi-agent systems application. The goal is to enrich $\mathcal{F}_{or}\mathcal{MAAD}$ with a *foreground design* based on a semi-formal language and allowing the use of the MDA transformation techniques to automatically generate an executable code. For this purpose, we have adopt the Agent Modeling Language (AML) ([5], [6]) as the formalism for the models representing steps of the $\mathcal{F}_{or}\mathcal{MAAD}$ methodology.

In order to guarantee a formal verification of the design models, we propose to translate the AML models of the *foreground design* to a *background design* which

uses a formal language $\mathcal{T}_{emporal}\mathcal{Z}$ [7] that integrates linear temporal logic into the Z notation. The *background design*, enables us to use formal verification tools supporting raw Z notation, such as $Z/EVES$ [8], for verification purposes. Such tools allow us to perform syntax, type and domain checking of our specification and to reason about correctness by proving several properties. This *background design* is described in [9] and [10] allowing a formal verified specification of an agent based application.

This paper is structured as follows. Section 2 presents a fragment of the AML meta-model. Section 3 describes the models proposed to cover the different phases of the $\mathcal{F}_{or}\mathcal{MAAD}$ approach. The translation rules are presented in section 4. Tools developed to support the use of the *foreground design* are described in section 5. Section 6 concludes the paper and dress perspectives to our work.

2 AML Meta-Model

The Agent Modeling Language (AML) ([5], [6]) is a semi-formal visual modeling language for specifying, modeling and documenting systems that incorporate features drawn from multi-agent systems theory. It is specified as an extension to UML 2.0 in accordance with major OMG modeling frameworks (MDA, MOF, UML, and OCL). The current version of AML offers support for the abstraction of architectural and behavioral concepts associated with multi-agent systems, i.e. ontologies, MAS entities, social aspects, behavior abstraction and decomposition, communicative interactions, services, observations and effecting interactions, mental aspects used for modeling mental attitudes of entities, MAS deployment and agent mobility [5].

The AML meta-model is structured as packages according to the various aspects of MAS abstractions: mental package, architecture package, behaviors package, etc. In the reminder of this section, we will present, as an example, a fragment of the architecture package and the mental one.

The architecture package defines the meta-classes used to model architectural aspects of MAS, such as entities (agents, environment, ressources), social aspect, ontologies, etc. Fig. 1 is a fragment of the AML meta-model extracted from the architecture package. The mental package defines the meta-classes used to model mental aspects of MAS, i.e. mental attitudes of autonomous entities, which represent their believes and goals. It defines also meta-classes which can be used to model problem decomposition and complex problems, in particular representing intentionality in use case models and goal-based requirements modeling. Fig. 2 is a fragment of the mental package.

3 $\mathcal{F}_{or}\mathcal{MAAD}$: Towards a Model Oriented Approach for MAS Design

The $\mathcal{F}_{or}\mathcal{MAAD}$ approach is based on two main phases. The first one is a specification phase in which the user requirements are described. The second one is

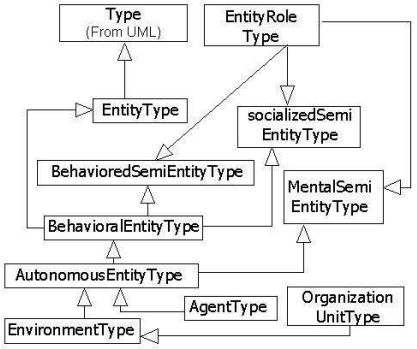


Fig. 1. An excerpt of the architecture package. [5]

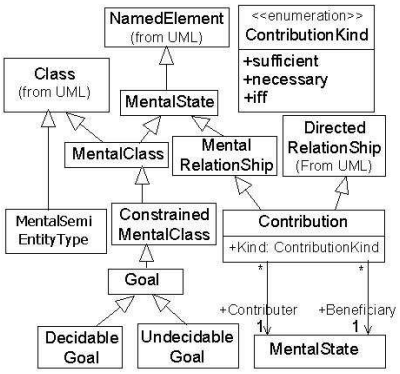


Fig. 2. An excerpt of the mental package. [5]

a design phase in which a detailed specification is derived based on a succession of refinements of collective (inter-agents) and individual (intra-agent) behaviors. In this section, we will review the $\mathcal{F}_{or}\mathcal{MAAD}$ steps and associate to them the corresponding models.

3.1 Specification Phase

In this phase, we specify the requirements which correspond, for a society of agents, to a common objective that must be achieved by these agents and the environment in which the agents evolve. This phase is captured by the requirement specification model in which, each entity is modeled by a class, a society of agent is modeled by an organization unit type, the environment is modeled by an environment unit type and an objective is modeled by a decidable goal. The common objective of an agent organization is expressed through a constraint associated to this organization.

As an example, the requirement specification diagram in the air traffic control application is illustrated by the Fig. 3. The class *System* models the agents's environment which is composed of an organization of planes (called *Planes*). This organization contains at least two planes and must achieve the goal *SolveConflict*. The note attached to the class *Planes* is a constraint that corresponds to the objective of the organization *Planes* which consists in solving each potential conflict situation between two planes.

3.2 Design Phase

The $\mathcal{F}_{or}\mathcal{MAAD}$ design process follows seven refinement steps. The first step defines a cooperation strategy for achieving the common objective. It consists in decomposing the common objective into a set of sub-goals, called local goals. The definition of an organization structure is performed into two steps. First,

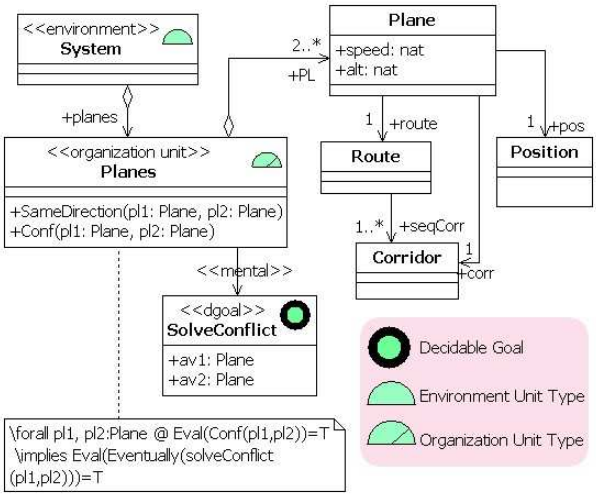


Fig. 3. A requirement specification model example.

we identify the roles by grouping local goals; then, we relate them with suitable relationships. Simultaneously, we assign roles to agents. The relationships between roles are translated at the agent level into organization links. Based on these links, we identify the needed collective behaviour. Finally, we have to define an appropriate individual behaviour for each agent. These steps will be given in details in the following sections.

Cooperation Strategy Definition This step consists in a decomposition of the common objective into local goals. In the *foreground design*, the cooperation strategy is defined onto two levels: the first one allows the description of the objective types and the decomposition relations between them using a class diagram; the second allows the instantiation of the objective classes using an object diagram. In the context of the air traffic application, the Fig. 4 shows the decomposition of the objective *SolveConflict*.

Organization Structure Definition The organization structure is depicted by the organization structure definition model composed of three diagrams: the role identification diagram, the organization structure diagram and the precedence order graph.

– Role identification diagram: this diagram describes the main roles needed to achieve the local goals. It is derived automatically from the cooperation strategy definition model by grouping local goals, instantiated from the same class, together. For example, the *negoWith*1 and *negoWith*2 goals (Fig. 4)

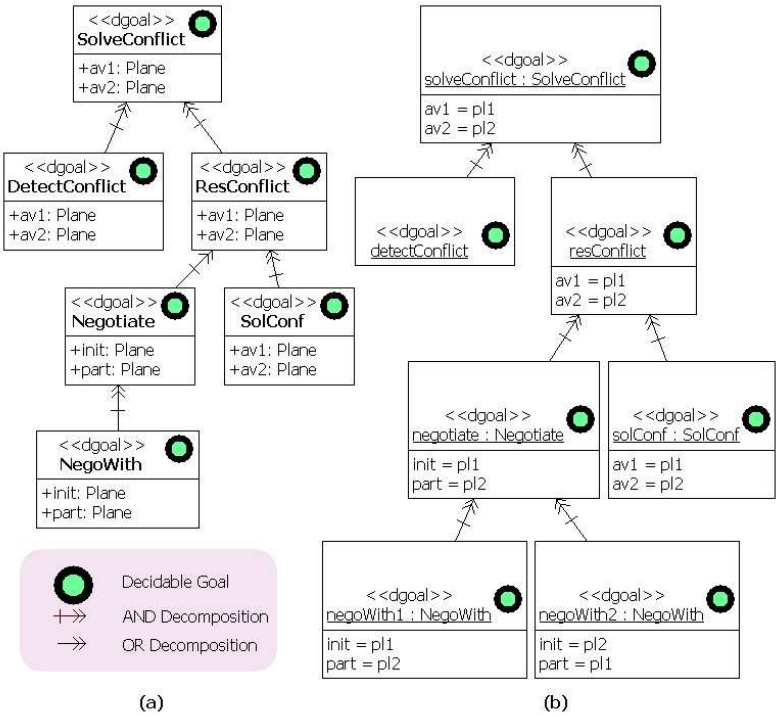


Fig. 4. A cooperation strategy definition model example: class level (a) and instance level (b).

are instances from the same goal class *NegoWith*. Thus, they are grouped to form the role *Negotiator* (Fig. 5).

– Organization structure diagram: this diagram is created by instantiating the roles identified in the previous diagram and creating the organization relations between them. In practice, we will identify common attribute values of the local goals of different roles. A common attribute between two roles indicates an organizational relation between them. As an example, the *solConf* local goal and the *negoWith*1 local goal have two common attribute values *pl*1 and *pl*2. Thus, an organization relationship between the *solver* role and the *negotiator* role will be created (Fig. 5). Given the set of roles and the set of relations between them, we will identify necessary agents and assign the retained roles to them. Given the set of agents and their corresponding roles, we can generate automatically the organization links between agents. In fact, each organizational relation between two roles leads to an organizational link between the agents having these roles. Fig. 5 shows an example of a complete organization structure diagram. It depicts three roles: *detector*, *negotiator*

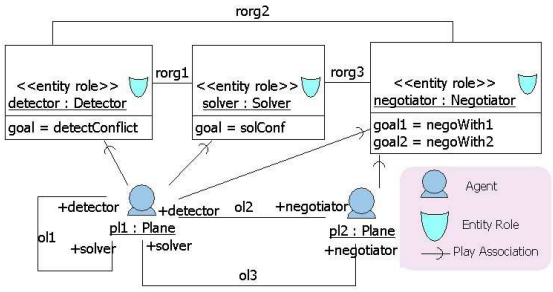


Fig. 5. An organization structure diagram example.

and *solver* with respective types *Detector*, *Negotiator* and *Solver*, two agents : $pl1$ and $pl2$, three organizational relations: $rorg1$, $rorg2$ and $rorg3$ and three organizational links $ol1$, $ol2$ and $ol3$.

– Precedence order graph: this activity diagram models the precedence order between local goals and serves to facilitate the identification of the necessary agents.

3.3 Collective Behaviour Definition

The collective behavior of the agents is defined according to their organizational links. In fact, an organizational link established between two agents leads to a sequence diagram describing the messages exchanged between them. In the air traffic control application, the collective behaviour definition model (Fig. 6) describes the negotiation protocol between two planes.

3.4 Individual Behavior Definition

The individual behavior definition model describes, with an activity diagram, the behavior of each type of agent. The agent actions and the sent messages are described with more details by a class diagram. Fig. 7 details some agent actions as behaviour fragments.

4 Translation to $\mathcal{T}_{emporal}\mathcal{Z}$

In order to verify some properties in our models, we propose to use formal verification techniques. The formal verification is applied after each step of the design methodology and is composed of two parts. The first part allows one to translate automatically the models to the $\mathcal{T}_{emporal}\mathcal{Z}$ formal language. The second one consists in verifying some theorems. The $\mathcal{T}_{emporal}\mathcal{Z}$ formal language was presented in [7] and is the result of the integration of the temporal formulas into Z schemas. This section presents the principals rules allowing translation and the theorems

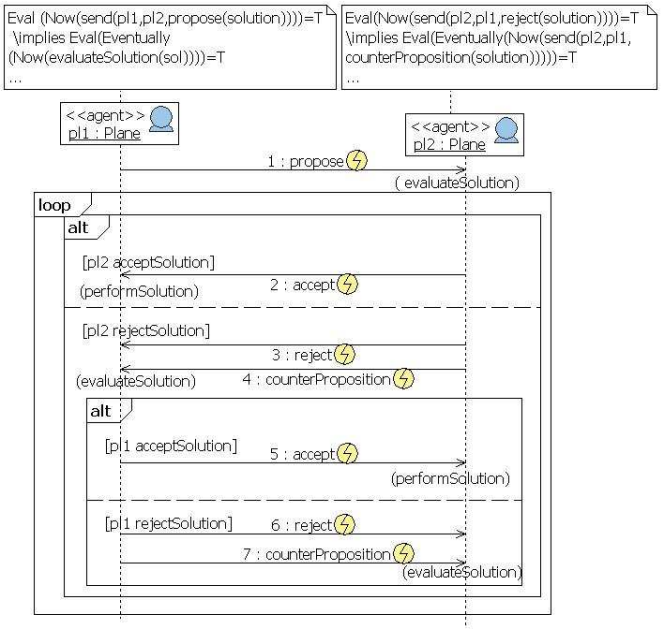


Fig. 6. A sequence diagram modeling the negotiation protocol.

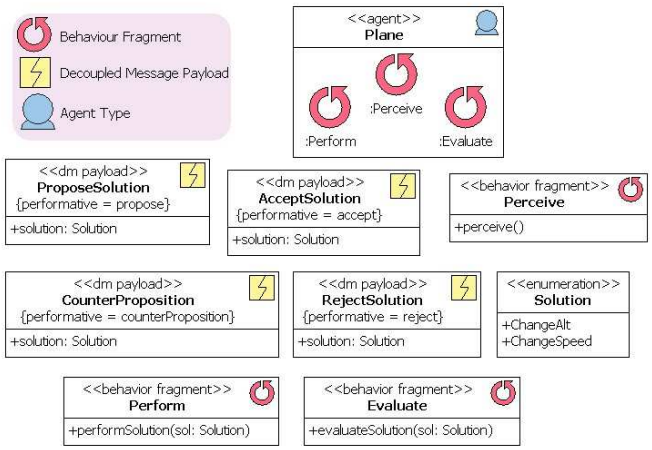


Fig. 7. An individual behaviour definition model.

to be proved. For space constraints, we show only some results of the translation of the *requirement specification model* and the *cooperation strategy definition* one. More details can be found in [9].

4.1 Translation of the Requirement Specification Model

The translation of this model allows one to define the Z schemas modeling the entities of the system and the environment of the agents. In the case of the air traffic control, the requirement specification model leads to a collection of Z schemas. For example, we mention the *Plane* schema describing a plane, the *Planes* schema representing an organization of planes and the *System* schema describing the environment.

$$\begin{array}{|l} \textit{Plane} \\ \hline \textit{speed} : \mathbb{N} \\ \textit{alt} : \mathbb{N} \\ \textit{route} : \textit{Route} \\ \ldots \\ \hline \end{array} \qquad \begin{array}{|l} \textit{Planes} \\ \hline PL : \mathbb{F}\ \textit{Plane} \\ \hline \end{array} \qquad \begin{array}{|l} \textit{System} \\ \hline \textit{planes} : \textit{Planes} \\ \hline \end{array}$$

4.2 Translation of the Cooperation Strategy Definition Model

The translation of the cooperation strategy definition model to $\mathcal{T}_{emporal}\mathcal{Z}$ completes the specification of type *Formula* [9] with the atomic formulas (local goals) and leads to the *implementation*0 schema including *System* schema and containing declaration of variable L having the type $\mathbb{F}$ *Formula* (Variable L is the set of the local goals). In addition, we generate the *CoopStrategy* theorem which guarantees that the common objective can be derived from the local goals (these concepts are described in [9]).

$$\begin{array}{l} \textit{Formula} ::= \\ \ldots \mid \textit{detectConflict}\langle\langle \textit{Plane} \times \textit{Plane} \rangle\rangle \\ \mid \textit{solConf}\langle\langle \textit{Plane} \times \textit{Plane} \rangle\rangle \\ \mid \textit{negoWith1}\langle\langle \textit{Plane} \times \textit{Plane} \rangle\rangle \\ \mid \textit{negoWith2}\langle\langle \textit{Plane} \times \textit{Plane} \rangle\rangle \end{array} \qquad \begin{array}{|l} \textit{implementation}0 \\ \hline \textit{System} \\ L : \mathbb{F}\ \textit{Formula} \\ \hline L = \{\textit{detectConflict}(av1, av2), \\ \textit{negoWith1}(\textit{init}, \textit{part}), \ldots\} \\ \textit{Eval}(\textit{solveConflict}(av1, av2)) = T \\ \Leftrightarrow \textit{Eval}(\textit{detectConflict}(av1, av2)) = T \\ \wedge\ \textit{Eval}(\textit{resConflict}(av1, av2)) = T \\ \ldots \\ \hline \end{array}$$

4.3 Translation of the Organization Structure Definition Model

As seen in the last section, the construction of the organization structure definition model requires four steps. After each step, we have to translate the corresponding part of the obtained model in order to prove the adequate theorem.

- Translation of the roles instances:
  this transition is done after the identification and the instantiation of roles. It leads to the generation of *implementation*1 schema describing the roles in terms of their local goals. It includes the *System* schema and contains the declaration of variable R typed $\mathbb{F}\, Role$ representing the system set of roles. Moreover, it allows the generation of *Completeness* theorem. This theorem states that each local goal belongs to a role, and the roles cover all local goals [9].
- Translation of organization relations between roles:
  this transition is done after the creation of the organization relation between roles. It allows the generation of the *implementation*2 schema describing the organizational relations in term of their participant roles. It includes the *System* schema and contains the declaration of the variable *Rorg* typed $\mathbb{F}\, OrgRelationship$ representing the set of organizational relations. We can also generate the *RoleParticipant* theorem verifying that the organizational relation participants cover the set of roles.
- Translation of the links with the stereotype *play*:
  this translation is done after the role assignment step. It leads to the generation of the *implementation*3 schema describing the roles of each agent according to a set of rules such as: "each object oA with the stereotype $<<$ *agent* $>>$ connected to an object oR with the stereotype $<<$ *entity role* $>>$ by a link with the stereotype $<<$ *play* $>>$ leads to a variable *oARoles* typed $\mathbb{F}\, Role$ in the *implementation*3 schema". The *RoleAssignement* theorem is also generated in order to verify that each role was assigned at least to one agent.
- Translation of organization link between agents:
  this transition is done after dressing the complete organization structure definition model. It allows the generation of the *implementation*4 schema describing the organizational links between agents. It includes the *System* schema and contains the declaration of the variable *organizationLink* typed $\mathbb{F}\, OrganizationLink$ representing the organizational links set.
  The *Instantiation*1 and *Instantiation*2 theorems are also generated stating that every organizational link instantiates an organizational relationship and that every organizationa relationship is instantiated by an organizational link.

4.4 Translation of the Collective and Individual Behavior Definition Model

The translation of these models to $\mathcal{T}_{emporal}\mathcal{Z}$ is done according to the following rules:

– Translation of the messages payloads:
classes with the stereotype << *dm payload* >> allow the description of messages exchanged between agents. Each one of these classes is characterized by a performative representing the message name and a list of attributes representing the transmitted objects. The messages payloads lead to the creation of a new type called *Message* whose values are the messages payloads performatives.

$$Message ::= propose\langle\langle Solution\rangle\rangle \mid accept\langle\langle Solution\rangle\rangle \mid reject\langle\langle Solution\rangle\rangle \mid counterProposition\langle\langle Solution\rangle\rangle$$

– Translation of the behaviour fragments:
the behaviour fragments describe the actions performed by an agent. Each behaviour fragment is characterized by a name and the set of actions. The behaviour fragments lead to the definition of a new type called *Action* whose values are the actions existing in these behaviour fragments.

$$Action ::= send\langle\langle Plane \times Plane \times Message\rangle\rangle \mid receive\langle\langle Plane \times Plane \times Message\rangle\rangle \mid perceive \mid performSolution\langle\langle Solution\rangle\rangle \mid evaluateSolution\langle\langle Solution\rangle\rangle$$

– Translation of temporal constraints:
send and receive actions are constrained by temporal constraints given by the user. These constraints will be inserted in the predicative part of the *implementation*6 schema. Thus, constraints specified in the diagram of Fig. 6 leads to the *implementation*6 schema. We also generate the *VerifSpec* theorem verifying that the obtained specification allows the achievement of the initial common objective.

5 $\mathcal{F}_{or}\mathcal{MAAD}$ Tools

StarUML [1] is a UML modeling framework supporting the MDA approach. This framework is characterized by its flexibility and its functionality extensibility. Thus, *StarUML* allows adding new functions in order to satisfy the user's requirements.

In order to be adapted to the $\mathcal{F}_{or}\mathcal{MAAD}$ approach, we propose to extend *StarUML* by (1) the insertion of a new approach in the approach part of *StarUML* called *ForMAAD approach* that integrates the presented models and that can be selected when launching *StarUML* ; (2) the extension of the UML profile part by creating a new profile called $\mathcal{F}_{or}\mathcal{MAAD}$ that is a part of the AML profile allowing the modification of the tool palette content for each diagram of the *ForMAAD approach*; the addition of some *JScript* scripts allowing the automatization of the model generation; (3) the addition of the $Add - In\ COM$ object developed under the *NetBeans* environment and allowing the transformation of

[1] http://staruml/sourceforge.net

the generated models into LaTeX; and (4) the insertion of a new panel called $\mathcal{F}_{or}\mathcal{MAAD}$ (figure 8) integrating some commands assisting the user to move from one $\mathcal{F}_{or}\mathcal{MAAD}$'s step to another and translating the resulting models into $\mathcal{T}_{emporal}\mathcal{Z}$.

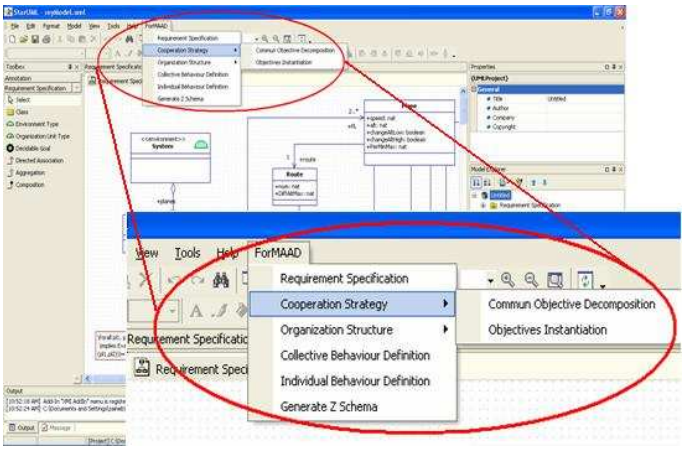


Fig. 8. $\mathcal{F}_{or}\mathcal{MAAD}$ menu.

The generation of a LaTeX file from the $\mathcal{F}_{or}\mathcal{MAAD}$ models follows two steps. The first step allows the creation of a *UML XMI* file using the transformation tool proposed by *StarUML*. The second step, consists on the application of a set of transformations of the *UML XMI* file in order to generate the LaTeX file. These transformations implement the translation rules presented in the previous section using some *XSLT* (eXtensible Styles Language Transformation) programs. Thus, the LaTeX file presenting a formal specification of the designed application, can be imported by the *Z/EVES* tool in order to prove the necessary theorems and to guarantee the requirement satisfaction.

6 Conclusion

In this paper, we proposed a reformulation of the $\mathcal{F}_{or}\mathcal{MAAD}$ methodology in terms of the MDA paradigm. Our main contribution consists in providing a set of methodological hints which guide the design process and stressing the correctness of the obtained design with respect to the requirements specification. Thus, we defined two ground designs: in the *foreground design*, we present a model oriented representation using AML and in the *background design*, we propose the translation of each model of the *foreground design* into a formal language called $\mathcal{T}_{emporal}\mathcal{Z}$ ([7]) that consists in the introduction of a temporal operators in the *Z* schemas enabled us to make use of *Z* supporting tools, like *Z/EVES*

[8], for syntax and type checking, as well as reasoning about the correctness of refinement steps.

As an example, we cited the air traffic control problem that allowed an illustration of the proposed method. Indeed, we designed a decentralized agent-based solution for conflict control in air traffic. The solution models a plane as an autonomous agent able to detect potential conflicts. The effective resolution of a conflict is the result of a negotiation process between planes.

The presented design process is supported by extending the *StarUML* tool in order to define a $\mathcal{F}_{or}\mathcal{MAAD}$ profile and to integrate the proposed approach with the five models. In this tool, we implemented the necessary rules allowing the transition into $\mathcal{T}_{emporal}\mathcal{Z}$ and the generation of the required theorems in order to be proved with the $Z/EVES$ tool.

It is necessary to point out that these results, though original and promising, constitute a first step in the development process of MAS. Thus, our perspective consists in the pursuit of the proposed process in order to define a complete model oriented approach allowing the code generation of the designed system starting from the verified abstract specifications generated by $\mathcal{F}_{or}\mathcal{MAAD}$.

References

1. Pavon, J., Gomez-Sanz, J.J., Fuentes, R.: The INGENIAS Methodology and Tools. In: Agent-Oriented Methodologies. (2005) 236–276
2. Guessoum, Z., Jarraya, T.: Meta-Models and Model-Driven Architectures. In: the AOSE TFG AgentLink3 meeting, Ljubljana (2005)
3. Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented Modelling. In: Proceedings of 6th International Workshop AOSE 2005, Utrecht (2005)
4. Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M.P.: Model Driven Engineering for Designing Adaptive Multi-Agent Systems. In: International Workshop on Engineering Societies in the Agents World (ESAW), Athens, Greece (2007)
5. Technologies, W.: AML 0.9 AML Specification. Technical Report 2004–12–20 (2004)
6. Trencansky, I., Cervenka, R.: Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS. Informatica **29** (2005) 391–400
7. Regayeg, A., Hadj-Kacem, A., Jmaiel, M.: Specification and Verification of Multi-Agent Applications using Temporal Z. In: 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004), 20-24 September 2004, Beijing, China, IEEE Computer Society (2004) 260–266
8. Meisels, I., Saaltink, M.: The Z/EVES 2.0 Reference Manual. Technical Report TR–99–5493–03e, ORA, Canada (1999)
9. Hadj-Kacem, A., Regayeg, A., Jmaiel, M.: ForMAAD: A Formal Method for Agent-Based Application Design. Journal of Web Intelligence and Agent Systems **5** (2007) 216–334
10. Regayeg, A.: Approche Formelle de Développement de Systèmes Multi-Agents : de la Spécification à la Conception. PhD thesis (2009)
11. Regayeg, A., Kallel, S., Hadj-Kacem, A., Jmaiel, M.: ForMAAD Method: An Experimental Design for Air Traffic Control. International Transactions on Systems Science and Applications **1** (2006) 327–334

An architectural perspective on multiagent societies*

Juan Manuel Serrano and Sergio Saugar

University Rey Juan Carlos, Madrid, Spain
juanmanuel.serrano@urjc.es sergio.saugar@urjc.es

Abstract. This paper attempts to explicate multiagent societies by exploiting the notion of software connector. From this architectural perspective, the social environment is structured in terms of a tree of *social interactions*, viz. a composite connector which provides the context for *speech acts*, *invocations* and *observations*, the three atomic interaction mechanisms provided by the environment. The paper also shows how to configure these social connectors for a particular application using a UML profile. This overall approach is intended to achieve a better alignment of multiagent societies with mainstream software engineering, as well as challenge some common architectural assumptions, namely the application-independence of software connectors.

1 Introduction

Arguably, the most salient and distinctive feature of multiagent societies is represented by the *social environment*, viz. the software infrastructure that mediates the interactions among agents and provide them with access to different types of resources. The social environment plays a role akin to the one played by middleware in mainstream software engineering [11]. Unlike conventional middleware, however, the social environment stands as a first-class design abstraction for application developers [12].

Software architecture is a mature software engineering discipline [2, 1] that also places significant importance to the separation of concerns between interaction and computation. In fact, component interactions are embodied in first-class abstractions, namely *software connectors* [3]. Although multiagent systems have already been approached several times from an architectural perspective [10, 9, 5], the notion of software connector has largely been omitted.

This paper attempts to alleviate this omission by providing a connector-based account of social environments. Thus, section 2 puts forward four kinds of *social connectors*, namely *speech acts*, *invocations*, *observations* and *social interactions*, which represents different kinds of interactions between agents and resources through the social environment. From this perspective, programming the social environment becomes a matter of defining the *types* of social connectors that conforms to the application domain. Section 3 introduces an UML profile for the design of these social connector types. The paper concludes with a discussion on related and current work.

* Research sponsored by project TIN2006-15455-C03-03

2 C&C architecture of multiagent societies

According to the catalogue of architectural viewtypes proposed in [1], *modular* views focus on the implementation units of the system and their relationships (functional dependencies, inheritance and part-of relations, etc.), whereas *Component & Connector* (C&C) views describe the structure of the system from a runtime perspective, thus focusing on the units of execution (components) and interaction (connectors). An *architectural style* of the C&C viewtype, such as object-oriented, pipe-and-filter, publish-subscribe, etc., is defined through different types of components and connectors, and hence characterise a particular kind of computational model. A crucial feature in the definition of a new type of connector consists of the roles played by their participating software components. Thus, invocation connectors involves two components that play the *caller* and *callee* roles.

Middleware infrastructures and C&C styles are tightly related. In fact, middleware infrastructures are *composite connectors* which offer several kinds of atomic interaction mechanisms to distributed components. For instance, CORBA-based middleware offers several variants of invocations, the characteristic connector of the object-oriented style: synchronous, deferred synchronous, one-way method calls, etc. Being connectors, middleware infrastructures are characterised by a number of role types. For instance, in object-oriented middleware, the major role played by interacting components is that of *object*. Thus, a CORBA software component (e.g. programmed in Prolog) is not an object due to certain intrinsic properties that it posseses, but because it is attached to an ORB to provide the services specified by its IDL specification. Indeed, in a distributed setting objects are essentially roles, not components.

Since multiagent societies are distributed systems, the C&C perspective leads us to explicate the nature of these kinds of systems in terms of the types of connectors supported by social middlewares and the roles played by software components interacting through them. Figure 1 describes schematically the C&C structure of multiagent societies proposed in this paper.

2.1 Social components

There are two kinds of components interacting through a social middleware: *agents* and *resources*. Agent and resource roles are represented in figure 1 as stick figures and triangle icons, respectively. Thus, the only agent components are c_2, c_3 and c_4, and the resource roles are played by components c_1 and c_5. Components behaving as agents are characterised by the goal that they purport to achieve within the social environment, and are regarded as heterogeneous and autonomous, i.e. they can neither be forced to act nor its state be altered. Agent roles also hold an event mailbox which stores any notification addressed to them by the environment. Unlike agents, resources are non-autonomous, and provide different computational or informational (e.g. virtualization [11]) services to the multiagent society where they are attached. Basically, resources are akin to objects deployed in a social setting.

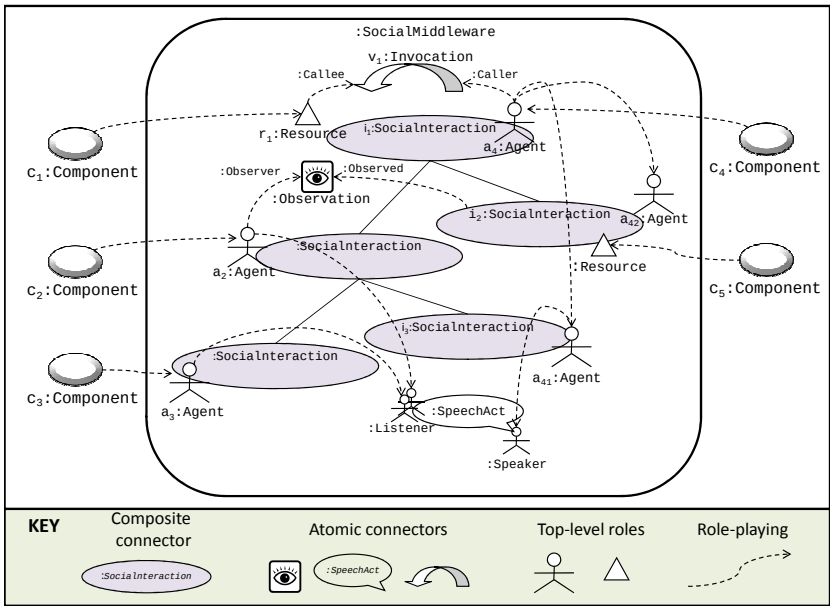


Fig. 1. Schematic C&C view of multiagent societies

2.2 Social connectors

In order to achieve their purpose, agents must be able to say things (possibly to other agents), manipulate the resources and observe their environment. Accordingly, three kinds of atomic interaction mechanisms are proposed: *speech acts*, *invocations* and *observations*. Besides, a composite social connector, namely *social interactions*, is provided which enables structuring the interaction space of the social environment.

Being connectors, atomic interaction mechanisms are stateful entities, not events (e.g. transient messages delivered through the low-level messaging infrastructure), and they are characterised by specific roles (e.g. the *speaker* and *listener* roles of speech acts). The initiation of these interactions is governed by *empowerment* rules, and their finishing by *permission* norms. Thus, if some agent is institutionally capable to say something (i.e. if it is empowered), the speech act will be initiated. Then, if the circumstances for saying that are met (i.e. if the speech act is permitted) the illocutionary purpose of the speech act will be brought about (i.e. the addressees will be notified, institutional facts will be created, etc.). Statefulness is also a necessary requirement for frozen interactions whose execution is neither permitted nor prohibited [6]. Moreover, it also allows for speech acts to be performed in a synchronous modality, meaning that addressees must listen to the speech act in order for the interaction to proceed. Figure 1 shows that agent a_{41} has said something to agents a_2 and a_3, who has listened to the speech act.

Social interactions represent social processes of different kinds and scales (e.g. conversations, teams, groups, organizations, etc.), and provide the context for the activities of agents and resources. In fact, *agents* and *resources* represent the two types of roles of this type of social connector. Social interactions can be decomposed into lower-level subinteractions, in such a way that the whole interaction space of the multiagent society is structured in terms of a tree of social interactions. Besides this connector hierarchy, the topology of the multiagent society also consists of the *role-playing* agent hierarchies. These runtime structures represent the decomposition of the agent activity according to the different contexts in which it participates. Thus, the activity of agent a_4 within the context of social interactions i_2 and i_3 is represented by agents a_{41} and a_{42}, respectively. The topology of the interaction space evolves dynamically according to the initiation and finishing of social interactions, which may happen in one of two ways: "automatically", according to the general rules of the society, or "manually" through the standard speech act declarations *SetUp* and *Close*.

3 UML Profile for social connectors types

Taking into account section 2, programming the social environment amounts to declaring the types of social connectors which implement the functional requirements of the application domain – as far as interaction is concerned. For instance, *program committees*, *submissions*, *reviewing teams*, etc., are common social interaction types of a conference management application; *submit* a paper and *notify* its acceptance or rejection are among its characteristic speech act types; last, *observe a review* is a common type of observation which is characterised by specific empowerment and permission rules (e.g. permission is granted to authors during the rebuttal stage). Note that some social connector types may be largely generic and, hence, reusable across many applications. For instance, the design of the reviewing team may profit from customizing a generic *discussion group* social interaction type.

The definition of social connector types requires a metamodel which identifies the programmable features of social interaction, speech act, observation and invocation connectors. This section briefly describes a light-weight implementation of this metamodel in terms of an UML Profile, and its application to the conference management domain (cf. figure 2). Note that the UML diagram of figure 2 actually represent a *modular* view of the application, in contrast with the C&C view shown in figure 1. As figure 2 shows, social interaction types are defined by stereotyped *use cases*. This is in accordance with the UML standard, since social interaction types can be regarded as units of functionality offered by the social middleware (the system) to its external software components (the users). Types of agents and resources are represented as stereotyped UML actors (in the latter case, using a distinguishing icon). This decision is also consistent with the UML metamodel which defines actors as *roles* played by external entities in its interaction with the system.

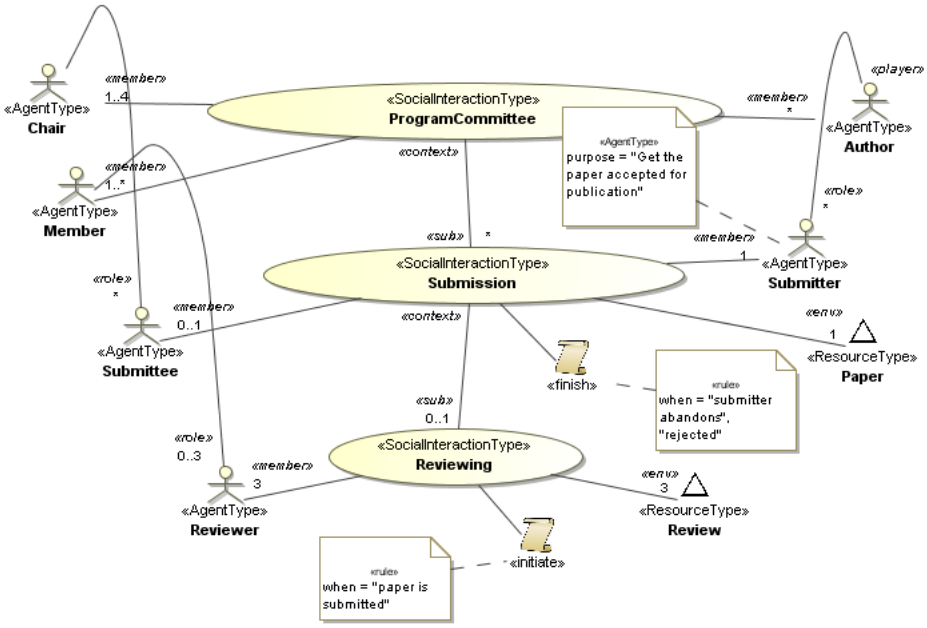


Fig. 2. Social interaction types of the conference management application

Lack of space precludes a detailed description of the stereotypes and tagged values which specialises the use case metamodel for representing social interaction types. Essentially, these allow the designer (1) to constrain the types of subinteractions in which its activity is decomposed, as well as their types of roles: member agents and environmental resources («context», «sub», «member», «env»); (2) specifying the agent role types («purpose» and «role»); and (3) declaring their life-cycle rules («initiate» and «finish»). Thus, reviewing interactions can only take place within the context of submissions, are automatically initiated when the paper is submitted, and provides the context for three reviewer agents and their corresponding reviews.

4 Discussion

The connector-based conceptualization put forward in this paper facilitates the comparison of multiagent societies and traditional architectural styles. For instance, in terms of the connector taxonomy proposed in [3], *speech acts*, *invocations* and *observations* are *communication* connectors, since they support the transmission of data among components (namely, agent and resource components). Moreover, invocations also fulfill a *coordination* role. They differ from common message passing, data access and procedure call connectors in the rules which govern their life-cycle, viz. empowerment and permission rules. Besides, they take place within the context of social interactions, a *facilitation* connector

which helps to structure the interaction space of the social environment. Some types of speech acts, particularly declarations (e.g. *SetUp* and *Close*), may also play a facilitator role. The connector-based approach has also inspired distinguishing features such as the synchronicity of speech acts and the possibility of frozen executions [6]. Similarly, the accompanying role-based notion of agenthood promotes higher levels of openness, autonomy and heterogeneity of software components.

Our approach fully endorses the view of the agent environment as a first-class design abstraction [12], and shows how it can be realised through a social connector metamodel. As such, this overall approach challenges the common view in the software architecture community which regards connectors as application-independent architectural elements [3]. In this regard, our proposal is closely aligned to the notion of *coordination artifact* [4].

The work presented in this paper attempts to provide the software architectural foundations of an ongoing research effort aimed at the design of a social interaction programming language. The runtime semantics [5, 8, 6] and type system [7] of this language, named SPEECH, provides the formal semantics for the social connectors and UML profile presented in this paper.

References

1. Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 2nd edition, 2003.
2. Paul C. Clements and Mary Shaw. "the golden age of software architecture" revisited. *IEEE Software*, 26(4):70–72, 2009.
3. Nikunj R. Mehta, Nenad Medvidovic, and Sandip Phadke. Towards a taxonomy of software connectors. In *ICSE*, pages 178–187. ACM Press, June 2000.
4. A. Ricci and M. Viroli. Coordination artifacts: A unifying abstraction for engineering environment-mediated coordination. *Informatica*, 29:433–443, 2005.
5. Juan Manuel Serrano and Sergio Saugar. Operational semantics of multiagent interactions. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *AAMAS'07*, pages 889–896. IFAAMAS, 2007.
6. Juan Manuel Serrano and Sergio Saugar. Dealing with incomplete normative states. In *Proc. COIN@MALLOW workshop. Revised and selected papers.* Springer, 2009.
7. Juan Manuel Serrano and Sergio Saugar. Programming social middleware through social interaction types. In *LADS'09. Revised and selected papers.* Springer, 2009.
8. Juan Manuel Serrano and Sergio Saugar. Run-time semantics of a language for programming social processes. In Michael Fisher, Fariba Sadri, and Michael Thielscher, editors, *CLIMA IX*, volume 5405 of *LNAI*, pages 37–56. Springer, 2009.
9. Munindar P. Singh and Amit K. Chopra. Programming multiagent systems without programming agents. In *Proc. of the AAMAS ProMAS workshop*, 2009.
10. Danny Weyns. Special issue on multiagent systems and software architecture. *IJAOSE*, 2(1), 2008.
11. Danny Weyns, Alexander Helleboogh, Tom Holvoet, and Michael Schumacher. The agent environment in multi-agent systems: A middleware perspective. *Multiagent and Grid Systems*, 5(1):93–108, 2009.
12. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *JAAMAS*, 14(1):5–30, 2007.

Developing an Agent Systems Reference Architecture

Duc N. Nguyen[1], Robert N. Lass[1], Kyle Usbeck[1], William M. Mongan[1], Christopher T. Cannon[1], William C. Regli[1], Israel Mayk[2] and Todd Urness[2]

[1] Applied Communications and Information Networking Institute, Drexel University
{dn53, urlass, kfu22, wmm24, ctc82, regli}@cs.drexel.edu
[2] Communications-Electronics Research, Development and Engineering Center, US Army

Abstract. One reason for the slow adoption in industry of agent-oriented methodologies as a paradigm for developing systems is the lack of integration and general-purpose technologies. To this end, there is a need to define common patterns, relationships between components, and structural qualities of an agent system. A reference architecture for agent-based systems would suit this need. This work describes the methodology for constructing an agent systems reference architecture by combining reverse software engineering techniques and tools and a documentation methodology. The goal of the resulting reference architecture is to identify common patterns and relationships between concepts present in agent systems to aid in describing and designing new agent systems.

1 Introduction

Using agent-based approaches for constructing large complex distributed systems can provide advantages over traditional methods. Unfortunately, industry has been slow to adopt this agent-oriented paradigm. One reason for this slow adoption is the lack of integration and general-purpose technologies [7]. Standards bodies such as the Foundation for Intelligent Physical Agents (FIPA)[3] are leading efforts to standardize protocols and formats of an agent-based system. However, there is a need to construct a reference *architecture* that defines the relationships between standardized terms and concepts of an agent-based system. Furthermore, such an architecture would give a set of architectural blueprints and best practices to aid in developing new agent frameworks and systems. To this end, a reference architecture for agent-based systems would speed other standardization efforts and adoption as a viable systems engineering perspective.

The purpose of the Agent Systems Reference Architecture (ASRA) is to describe relationships and structural qualities to support the construction of an agent-based system. The ASRA is created from multiple cross-cutting levels: the framework level, the system behavior level, and agent systems in the context of larger external systems. Constructing such an architecture for general systems is impossible given the various kinds of agent systems. One approach for building a reference architecture would be to analyze the tools used to build agent systems to determine the interactions between the functional concepts of an agent system.

This paper focuses on the process of developing a reference architecture for agent-based systems. Our approach is the application of a modified 4+1 View Model [3] to

[3] http://www.fipa.org

existing agent framework implementations creating five architectural views. We apply this process to functional concepts in an agent system to obtain the reference architecture.

We analyzed three existing agent framework implementations: Cougaar, JADE, and AGLOBE. This methodology is applied to agent frameworks rather than deployed agent systems because the functional concepts defined in the Agent Systems Reference Model (ASRM) [5] are already implemented in these frameworks.

The main contribution of this paper is a methodology for creating an agent systems reference architecture through the application of reverse engineering methodologies combined with the modified 4+1 View model used for documenting existing agent frameworks.

The rest of this paper is organized as follows: The next section provides a summary of related efforts in reference architectures for agent-based systems, and defines the terms *architecture*, *reference architecture* in the context of agent systems and agent frameworks. Section 3 describes the Agent Systems Reference *Model* and its basis for creating the ASRA. Section 4 describes how the 4+1 Model is applied to agent frameworks. Section 5 demonstrates the application of the process to create a portion of the ASRA. Finally, we conclude with a roadmap of continuing work for developing a reference architecture.

2 Background and Related Work

There is no general consensus for the definition of a reference architecture; however, we describe related standards efforts and reference architectures in this section.

The Foundation for Intelligent Physical Agents There are efforts to describe the behavior and interaction of agent systems. The FIPA Abstract Architecture Specification [2]discusses agent system architecture in an effort to promote interoperability and reusability. FIPA intends to provide a generic view on agent systems and describe how specific functionality should interact. This specification states low-level details such as the mechanisms for how agents perform service look-ups. The ASRA also focuses on identifying architectural paradigms and patterns in agent frameworks. So the ASRA acknowledges that service look-ups may be implemented in different ways and identifies the different ways of performing this action.

Reference Architecture for Situated Multiagent Systems Weyns *et al.* [6] developed a reference architecture for situated multiagent systems consisting from an agent and an application environment viewpoints. This architecture was developed through an interative process of analysis and validation studying different agent-based systems. In their reference architecture, the authors constructed multiple documents from different views: the module decomposition, the shared data, and the communicating processes views.

This reference architecture for situated multiagent systems has a similar result of constructing multiple documents for each view; however, the approach is different. The ASRA uses a combination of static and dynamic code analysis of representative agent framework implementations , whereas the former reference architecture examines several multiagent systems implementations.

RCS and RCS Related Work The Real-time Control System, first developed by Barbera, *et al.*at the National Institute of Standards and Technology, is a reference architecturefor hierarchical intelligent control. The RCS reference architecture provides for intelligent control what the ASRA provides for agent systems. As stated by the RCS reference architecture [1]: "the evolution of the RCS concept has been driven by an effort to include the best properties and capabilities of most, if not all, of the intelligent control systems currently known in the literature, from subsumption to SOAR, from blackboards to object-oriented programming."

The notion of a reference architecture has different meanings based on the viewpoints and concerns of the stakeholders. In this work, a reference architecture for agent-based systems is defined as a set of documents addressing patterns and component relationships of the functional concepts set forth in the ASRM.

3 The Agent Systems Reference Model

The Agent Systems Reference Model (ASRM) [5] is a model for software systems composed of agents. It establishes terms, concepts and definitions needed for the comparison of agent systems. The ASRA is an elaboration of the ASRM since it establishes relationships between concepts in agent frameworks and defines structural patterns for those concepts.

The ASRM defines an intelligent agent—or simply agent—as *situated* computational processes that embody one or more of the following qualities: autonomy, proactivity, interactivity, continuous, sociality, and/or mobility. The ASRM also formalizes concepts and layers of organization in an agent-based system. The layers described are: *agents*, *frameworks*, *platforms*, *hosts*, and *environments*. An agent-based system is the set of frameworks, the agents that execute in them, the platform (other software) that supports them and the hosts (hardware) upon which they execute.

The functional concepts of an agent system support overall system execution. They are essential in the definition of the ASRA and are made up of the following: Agent Administration, Security and Survivability, Mobility, Conflict Management, Messaging, Logging, and Directory Services.

4 Methodology for creating an Agent Systems Reference Architecture

Our approach to constructing a reference architecture for agent systems is to create multiple architecture documents by analyzing existing open source agent framework implementations and applying a rigorous 4+1 view model augmented with reverse engineering data.

Deriving 4+1 Views Using Reverse Engineering The 4+1 View Model [3] creates different architectural descriptions, or *views*, of software systems for different interested parties (*e.g.*, system developers, business-persons, customers). Each view identifies and describes the relationships between components and concepts. Interested parties will

view these relationships with different weights and significance, in some cases they are meaningless. The Views of the 4+1 model, and their construction for the ASRA, are described in detail in Section 5.

Our approach in deriving and documenting each architecture is a modified version of the 4+1 approach. Here, we begin by iterating over the functional concepts of the ASRM, and developing the Scenario View consisting of use cases. For each use case, we developed an agent that exercised the scenario, and performed reverse-engineering runtime analysis to obtain the Process View. This data provides a slice of the program (and, thus, the framework and other libraries), through which we can focus our static analysis in the Development View. Finally, this is abstracted into components using clustering algorithms to form the basis of the Logical View. By contrast to traditional 4+1 approaches, we document the most abstract views first and augment each with reverse engineering data or domain knowledge to create the next view.

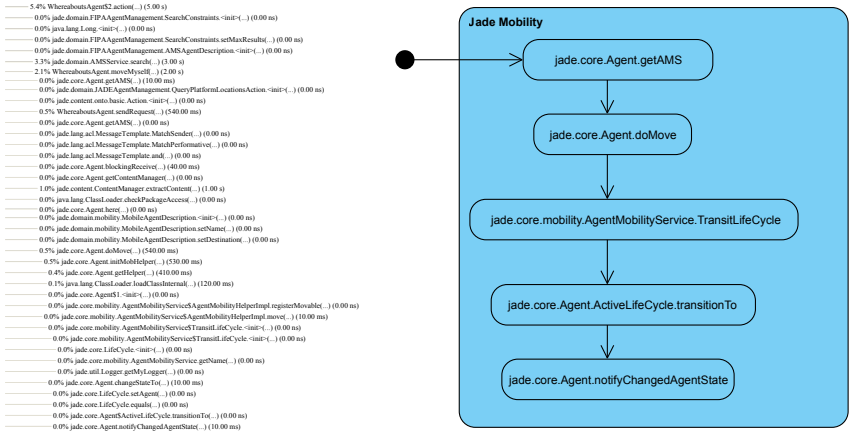
We utilized reverse software engineering tools and performed dynamic and static analysis [4] to assist with our data mining and move from scenarios and existing implementations to a reference 4+1 architectural description. In this approach, the data used to construct a view is used to inform the construction of the next view.

5 Case Study: JADE Mobility

We demonstrate the analysis and application of the modified 4+1 documentation model for agent mobility within an agent system implemented using the JADE framework.

The Scenario View: The documentation process begins with stating the use cases for the functional concept defined by the ASRM and further elaborating actors and invocation points. The intended audience are high-level practitioners who need explanation of concepts for an agent-based system.

The Process View: To create the Process view, dynamic analysis data is generated for a system by running a slice of a program representing the scenario. We create a process diagram to illustrate the process view. Reverse engineered runtime data augments the process diagram to create a package diagram to provide a conceptual view. For agent mobility, we generate a runtime trace by running a profiler against code snippets that demonstrate agent mobility. The runtime trace documents the percentage of time methods are invoked during the code's execution. Figure 1(a) displays the temporal view of a scenario demonstrating the invocation points of the agent mobility functional concept. The resulting process diagram after augmenting with package names is shown in Figure 1(b). Upon creating similar diagrams for AGLOBE and Cougaar, two patterns for defining the process of agent mobility emerge: *serialization mobility* and *shared-object mobility*. With serialization mobility as exhibited by JADE and AGLOBE, (1) the agent's thread of execution is paused, (2) the agent is converted into a transferable form, (3) the is transfered to the target platform, (4) the agent is converted back into an executable form, and (5) the agent resumes (or begins) thread of execution. With shared-object mobility (as is the case with Cougaar) mobile agents are shared between platform containers. The agent's state includes current platform location and



(a) Runtime Trace. (b) Process View Diagram

Fig. 1. Jade Mobility runtime trace and resulting Process view diagram.

upon "transmission", the agent's shared object state is modified to indicate the agent's new platform location.

The Development View: The development view is the static view of the agent system derived through the use of static code analysis tools and temporal data from the previously created process view. Static analysis tools produce a graph of all the software components. This data is analyzed and informed by the runtime analysis (to focus the search of a large static analysis data set) obtained during the construction of the Process View to create the Development View. This view provides a sense of the topology of the agent system software, including the logical components responsible for performing tasks and design patterns representing the connections between them. The resulting agent mobility functional concept is described using the data obtained in the serial approach, and consists of: **Destination Platform Discovery**, in which Directory Services are used to locate the addressing information of the destination platform, **Agent Encapsulation**, in which the mobile agent is serialized into a message from the Messaging functional concept, **Message Communication**, in which the agent is delivered to its destination via the messaging component, and **Agent Extraction**, in which the agent is extracted at the destination platform from its serialized state.

The Logical View: The logical view is constructed by observing the clustered runtime data generated from our reverse engineering tools, and organizing the major objects into packages. Although the process and logical views concern themselves with concrete system details in an architecture, they are also helpful to express the high level packages and interacting components existing in an agent system. For agent mobility, the abstracted logical view illustrates that even though agent mobility can be implemented using the functional concepts for directory services and messaging, there is the additional requirement to identify the messages as encapsulated serialized agents.

The Physical View: The physical view is normally reserved by the 4+1 model for non-functional requirements of a system, including deployment and administration concerns and is developed independently of the serial approach that generated the scenario, process, development, and logical views because it deals with more of the physical aspects of the framework and the agent system.. Although the ASRA addresses some of these concerns, the physical view section primarily concerns itself with the physical aspects and design decisions in deploying an agent system in a given environment. Otherwise this functional concept is not abstractable, so it is omitted here.

6 Conclusion and Future Work

This paper described our approach for creating a reference architecture for agent systems and frameworks using the 4+1 View Model. We demonstrated this application on the Agent Mobility functional concept in each view. This approach satisfies practitioners at multiple levels: high-level, system designer, system architect, developer, system deployer. The ASRA provides architectural design paradigms for agent framework functional concepts defined by the ASRM. These serve as architectural blueprints for constructing new agent frameworks, or identifying the functional components required to construct new systems using existing frameworks. Further documentation of the functional concepts and their interactions at each view is in progress and a complete document is forthcoming. Further work also includes creating reference architectures focusing on the paradigms of agents and external systems outside the traditional agent-based system construct (for example, agents integrated with web services) and on agent societies and communities.

References

1. James Albus and G. Rippey. RCS: a reference model architecture for intelligent control. In *Proceedings of the From Perception to Action Conference*, pages 218—229, September 1994.
2. Foundation for Intelligent Physical Agents. Abstract architecture, December 2002. `http://www.fipa.org/specs/fipa00001/`.
3. P. Kruchten. Architectural blueprints—The "4+1" view model of software architecture. *IEEE Software*, 12(6):42–50, November 1995.
4. W. M. Mongan, C. J. Dugan, R. N. Lass, A. K. Hight, J. Salvage, W. C. Regli, and P. J. Modi. Dynamic analysis of agent frameworks in support of a multiagent systems reference model. *IADIS International Conference Intelligent Systems and Agents*, 2007.
5. W. C. Regli, I. Mayk, C. J. Dugan, J. B. Kopena, R. N. Lass, P. J. Modi, W. M. Mongan, J. K. Salvage, and E. A. Sultanik. Development and specification of a reference model for agent-based systems. *IEEE Trans. On Systems, Man, and Cybernetics, Part C*, 39(5):572–596, Sep. 2009.
6. D. Weyns and T. Holvoet. A reference architecture for situated multiagent systems. *Lecture Notes in Computer Science*, 4389:1, 2007.
7. D. Weyns, H. V. D. Parunak, and O. Shehory. The future of software engineering and multi-agent systems. *Special issue on Future of Software Engineering and Multi-Agent Systems, International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 2008.

A Middleware Model in Alloy for Supply Chain-Wide Agent Interactions

Robrecht Haesevoets, Danny Weyns, Mario Henrique Cruz Torres, Alexander Helleboogh, Tom Holvoet, and Wouter Joosen

Distrinet, Katholieke Universiteit Leuven, 3001 Leuven, Belgium
robrecht.haesevoets@cs.kuleuven.be

Abstract. To support the complex coordination activities involved in supply chain management, more and more companies have autonomous software agents acting on their behalf. Due to confidentiality concerns, such as hiding sensitive information from competitors, agents typically only have a local view on the supply chain. In many situations, however, companies would like to expand the view of their agents to share valuable information such as transportation tracking and service delays. Non of the participating companies, however, has enough knowledge or authority to realize such interactions in a controlled manner.
In this paper, we present an organization middleware that offers a collaboration platform and enables agents to interact across the boundary of local interactions. Policies and laws enable companies to define the scope of interactions of their agents and the restrictions on their exposed information. Using Alloy, we formally define the relation between the interactions offered by the middleware, the exposed information and the provided policies and laws. This allows us to guarantee a number properties which are of particular interest to companies using the middleware.

Key words: Organisations and institutions; Social and organizational structure; Verification of MAS

1 Introduction

In today's competitive and globalized market, streamlined collaborations between business entities are a necessity. In the DiCoMas project[1], a joint research effort with academic and industrial partners, we have been studying the use of agents for managing collaborations between business entities in the domain of supply chain management. A key objective of this project is to improve integration and collaboration among supply chain partners.

Due to company-specific restrictions, such as hiding sensitive data from competitors or having clients exchange pricing info with subcontractors, companies typically only allow their agents to participate in local supply chain interactions [10]. As a result, agents only have a local view on the supply chain. Nevertheless, in many situations companies would like to extend the view of their

[1] DiCoMas: Distributed Collaboration using Multi-agent System Architectures: http://distrinet.cs.kuleuven.be/projects/dicomas/index.html

agents and allow them to participate in supply chain-wide interactions in a controlled manner. Examples are tracking containers throughout the supply chain or monitoring problems such as delays outside the local view of agents.

A typical way to structure such interactions between agents is by means of roles and organizations [7, 1]. In previous work [13], we have presented an organization model for collaborative multi-agent systems. Although the model is relatively simple, it is powerful enough to model controlled supply chain-wide interactions. A subset of the model is shown in Fig. 1. The core abstractions of the model are organization, role, and capability. Organizations, defined as a set of roles, specify the boundaries in which controlled interactions can take place. A role represents a concrete participation in the organization. It defines the agents that have access to the organization, and it defines the capabilities these agents have in the organization. Each capability represents a concrete interaction ability relative to another role in the organization.

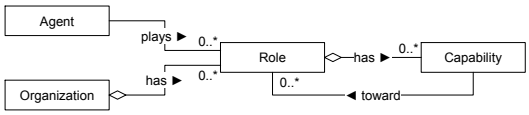


Fig. 1. A visual representation of the organization model.

Realizing organizations and managing their dynamics in a heterogeneous and distributed supply chain setting is a very complex task, for which none of the participating companies has enough authority or knowledge. Additionally, companies want guarantees before exposing confidential information or allowing their agents to collaborate with external parties.

To address these challenges we present an organization middleware approach. The middleware offers organizations and roles as a set of reusable programming abstractions to application developers. At run-time, the middleware realizes a collaboration platform. Agents provide the middleware with local information on the supply chain, and in return, the middleware offers managed organizations that enable agents to engage in supply-wide interactions in a controlled way. Companies can specify interaction laws to define the desired scope of interactions for their agents and a set of policies to restrict the information they expose, in order to deal with confidentiality concerns. These laws and policies will then be enforced by the middleware.

The use of organizational abstractions together with a middleware has a number of key benefits: (1) it allows to represent and structure supply chain-wide interactions at a high-level of abstraction; (2) it allows to separate the management of dynamic supply chain-wide interactions, performed by the middleware, from the actual functionality, provided by agents participating in the interactions; (3) it allows to accurately restrict the interactions between agents according to provided policies in terms of capabilities.

The contributions of this paper are:

1. We motivate and specify a set of concrete requirements for supply chain-wide interactions in the domain of logistics for supply chain management.
2. We present a formal model in the Alloy specification language [6] of an organization middleware supporting supply chain-wide interactions. The model formally defines the relation between supply chain-wide interactions enabled by the organizations offered by the middleware and the local supply chain information exposed by the agents and the provided policies.
3. We assert a number of relevant properties offering companies formal guarantees in terms of confidentiality using the model and the Alloy Analyzer.

Overview of this paper. Section 2 introduces a running example together with a set of requirements for supply chain-wide interactions. The organization middleware is presented in Sect. 3 and illustrated in the running example. Section 4 presents the middleware model in Alloy and shows how the Alloy Analyzer can be used to assert a number of properties. Finally, related work is discussed in Sect. 5, and Sect. 6 concludes and reflects on future work.

2 Logistics in Supply Chain Management

In the domain of supply chain management, companies usually outsource their logistic activities to one or more specialized third-party logistics providers (3PL). To integrate and streamline the operations of different 3PLs, an extra level of outsourcing can be introduced, called fourth-party logistics providers (4PL). Figure 2 shows an example of a hierarchical outsourcing structure in a supply chain, used as a running example in this paper. In the example, several companies collaborate to realize the logistic needs of company 0. Company 0 has an outsourcing contract with company 1, which as acts as a 4PL and integrates the services of two 3PLs, company 2 and 3. Company 2, in turn, has two additional subcontractors, company 4 and 5. In the example, company 3 is currently carrying a container of company 0, and company 4 and 5 are expecting a delay.

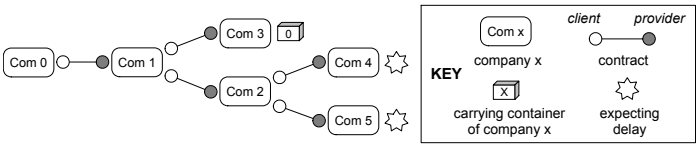


Fig. 2. Supply chain collaborations.

Due to confidentiality concerns, companies only allow their agents to participate in local interactions corresponding to active outsourcing contracts. As a result, agents only have a local view on the supply chain. Typical supply chain flows, such as information and services, are propagated through the supply chain based on local interactions. In the DiCoMas project, we aim to enhance the integration and collaboration of the supply chain partners to improve information

sharing and responsiveness. To realize this, agents acting on behalf of companies need extended views on the supply chain and have to interact across the supply chain in a controlled way. We give a number of concrete stakeholder requirements that motivate the need for supply chain-wide interactions. For clarity, the requirements are explained in the context of the running example.

Collaborative Planning. To create a planning in correspondence with the individual goals of each stakeholder, company 1 wants to use a collaborative planning approach. This requires agents of both clients, such as company 0, and subcontractors, such as company 2 and 3, to participate in coordinated planning and negotiation activities, while company 1 maintains a supervising position and can enforce the necessary restrictions on the involved interactions.

Traceability. Company 0 wants to track the location and status of its containers throughout the supply chain. Instead of having to contact its service provider, company 1, who in turn has to contact other service providers, company 2 or 3, and so on, company 0 requires it agents to directly interact with the agents of the current carriers of its containers, increasing responsiveness and reducing overhead. Using policies, intermediate companies such as company 1 should be able to restrict the information that can be exposed to company 0.

Improved Responsiveness in Case of Problems. As a 4PL, Company 1 wants its agents to be directly informed by agents managing third-party resources when serious problems occur, such as delays or decommitment. This enables company 1 to anticipate future problems at a supply chain-wide level and offer its clients a higher quality of service. Intermediate companies should be able to restrict the information exposed by their subcontractors.

3 The Organization Middleware

The previous section introduced a number of stakeholder requirements that underpin the need for supply chain-wide interactions. Such interactions can be modeled and coordinated using organizational abstractions we introduced in [13]. In this section we present an organization middleware that offers such organizations and roles as a set of reusable programming abstractions to application developers. At runtime, the middleware provides a collaboration platform and takes the responsibility of managing organizations and their dynamics, for which non of the partners in a supply chain has enough authority or knowledge.

Figure 3 gives a high-level overview of the approach. To participate, agents of supply chain companies have to provide the middleware with context information and a set of interaction laws. In return, the middleware offers agents a broader view on the supply chain and support for supply chain-wide interactions, while taking the responsibility of managing the interactions and their dynamics. Using a middleware allows us to separate the management of the organizations from the agents, who can now focus on realizing the functionality in organizations. Internally the middleware can be realized using different technologies including agents. Agents using the middleware have to conform to certain communication standards, which are outside the scope of the current model.

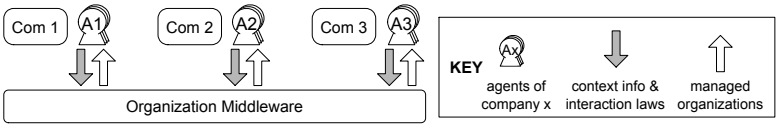


Fig. 3. High-level overview of the approach.

In the remainder of this section we first explain the notions of context and interaction laws in more detail. We then show how context and laws can be used by the middleware to offer organizations that enable controlled supply chain-wide interactions in the running example.

3.1 Context Information and Interaction Laws

Agents have to provide the middleware with local information on the supply chain, consisting of context and interaction laws. The completeness of the context depends on the amount of information exposed by the agents on behalf of the companies. Context includes information on companies, their dynamic properties, such as containers currently carried or expected delays, the current outsourcing contracts between companies, and a set of flow policies. Flow policies define the allowed supply chain flows between agents of particular companies. We currently consider two types of flows: information flow and service flow. These allow companies to specify which information exchange and which concrete service provision can take place between which specific companies. Flow policies are specified at the level of outsourcing contracts as allowed flows within outsourcing contracts as well as between different contracts. An example is shown in Fig. 4, illustrating how flow policies of different companies create a graph-like structure defining the allowed information and service flows at a supply chain-wide level.

Interaction laws allow companies to define in a declarative way the desired scope of the supply chain-wide interactions for their agents. In particular, an interaction law specifies a desired set of interaction partners whose agents should be allowed to participate in the interaction, such as "all providers of a company" or "all companies carrying a specific container", as well as the supply chain flows the interaction should enable between these partners.

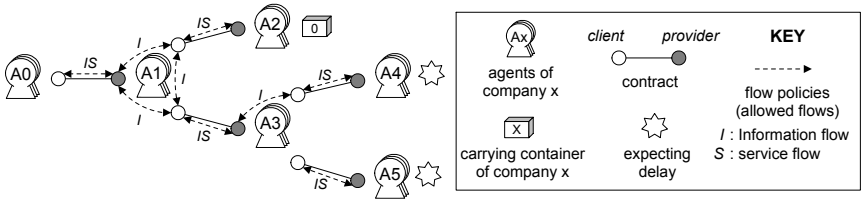


Fig. 4. Context consisting of flow policies and outsourcing contracts.

3.2 Realizing Supply Chain-Wide Interactions.

The middleware uses the interaction laws together with the current context to provide a set of organizations supporting the desired supply chain-wide interactions. Each organizations enables a set of interactions, defined by the capabilities of its role and each capability enables a specific supply chain flow toward another role in the organization in correspondence with the current flow policies. As context or laws change, the middleware adapts the organizations accordingly.

Figure 5 illustrates a set of organizations realizing the requirements for supply chain-wide interactions introduced in Sect. 2 for the running example. Organization 1 illustrates collaborative planning, enabling the agents of client 0 to exchange planning information with the agents of subcontractors 2 and 3. Role capabilities, compliant with the flow policies, show that company 1, as a 4PL, remains in a supervising position, ensuring clients have no capabilities to make any direct service requests to subcontractors. Organization 2 shows the tracking of a container throughout the supply chain, enabling the agents of company 0 to interact with the carrier of their container, the agents of company 3. Improved responsiveness is exemplified by organization 3, allowing agents of company 1 to interact with the agents of company 4, which is expecting a delay. Because company 2 wants to hide its internal outsourcing strategy, it does not allow any flows between company 5 and other parties, as illustrated in Fig. 4. As a result, company 5 is excluded from organization 3, although it is also expecting a delay.

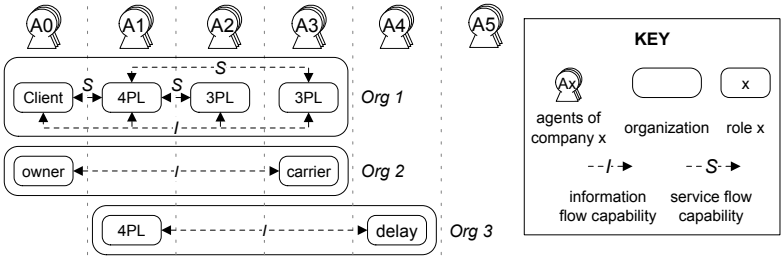


Fig. 5. Examples of organizations and roles realizing supply chain-wide interactions.

4 Middleware Model in Alloy

In this section we give a formal model of the middleware abstractions using the Alloy specification language. Alloy [6] is a structural modeling language based on first-order logic for expressing complex structural constraints and behavior in software systems. The Alloy Analyzer[2] is a constraint solver, supporting automatic simulation and checking of Alloy models within a specific scope. Simulation

[2] Alloy Analyzer 4 - `http://alloy.mit.edu/alloy4/`

consists of finding instances satisfying a specification, while checking consists of finding counter examples violating certain assumptions about a model. The Alloy analysis is based on the notion of *small scope hypothesis* [6], assuming that assertions checked within a well-chosen scope will also hold for larger scopes. However, with a well-chosen scope and model, it can even be possible to do a complete analysis for a specific setting.

The purpose of our formal model in Alloy is threefold: (1) present a rigorous specification of the main concepts of the organization middleware; (2) formally define which supply chain-wide interactions the middleware can and should provide, given the context and a set of interaction laws; (3) show how this model can be used together with the Alloy analyzer to guarantee a number of properties in terms of confidentiality constraints. Due to space constraints, parts of the formal specification are omitted. A complete model is available for download[3].

4.1 Middleware Model

The model is shown in Spec. 1. Concepts are represented by a number of *signatures*, each introducing a new set of atoms in the universe (*univ*) of the model.

Context Information. Context information consists of information on companies, their dynamic properties and their flow policies. We start by defining the signatures *Company* and *Contract* to represent companies and their outsourcing contracts. *Company* has one *field*, named *properties*, mapping each company to a set of properties, defined by the signature *Property*. *Contract* has three fields, two disjunct companies representing the client and provider in the contract, and a field *flows* mapping each contract to the set of supply chain flows that are allowed to take place within the contract. Supply chain flows are defined by the signature *Flow*. Subtypes *Info* and *Service* represent some of the typical supply chain flows, but more expressive subtypes can be introduced.

On line 11 the signature *context* defines the context of the middleware as a set of companies, contracts and flow policies. Flow policies are defined on line 14[4] as ternary relations which specify the allowed flows between different contracts. A signature fact on line 16[5] introduces an additional constraint to ensure companies can only define flow policies between their own contracts. We also define a help function *allowedFlows* on line 19[6] which returns the supply chain flows that are allowed between companies by the contracts and flow policies in the given context.

Interaction laws. Interaction laws are represented by the signature *Law* on line 25. The field *scope* specifies the desired scope of interaction, as the set of

[3] `http://www.cs.kuleuven.be/~robrecht/AOSE2010/`

[4] The field *flowPolicies* can refer to multiple flow policies. The Alloy syntax does not require the *set* keyword for relations.

[5] The *box join* `a[b]` is the equivalent of the relational join `b.a`. The `+` sign represents the union of two sets while the `&` sign represents the intersection.

[6] The set comprehension `{a: A | constraint}` returns all elements of A satisfying the given constraint. `*a` represents the reflexive transitive closure. `<:` and `:>` represent the domain and range restriction of a relation.

Specification 1 Middleware Model

```
sig Company{
  properties:set Property
}
sig Contract{
  disj client,provider:Company,
  flows:set Flow
}
sig Property{}
abstract sig Flow{}
one sig Info,Service extends Flow{}
sig Context{
  companies:set Company,
  contracts:set Contract,
  flowPolicies:Flow->contracts->contracts
}{
  all c1,c2:Contract | c1->c2 in flowPolicies[univ] implies
    some c1.(client+provider) & c2.(client+provider)
}
fun allowedFlows[context:Context]:Flow->Company->Company{
  {flow:Flow,com1,com2:Company | some c1,c2:context.contracts |
    flow in c1.flows & c2.flows and
    com1+com2 in (c1+c2).(client+provider) and
    c2 in c1.*(flows.flow<:context.flowPolicies[flow]:>flows.flow)}
}
sig Law{
  scope:Flow->Company->Company
}
fun propertyBasedSelection[p:Property, vp:Company, context:Context]:set Company{
  {c:Company | p in c.properties and Info->c->vp in allowedFlows[context]}
}
sig Role{
  company:Company,
  capabilities:Role->Flow
}
sig Organization{
  roles:set Role
}
fun enabledFlows[org:Organization]:Flow->Company->Company{
  {flow:Flow,com1,com2:Company | some r1,r2:org.roles |
    r1.company = com1 and r2.company = com2 and r2->flow in r1.capabilities}
}
sig MiddlewareModel{
  context:Context,
  laws:set Law,
  orgs:set Organization
}{
  enabledFlows[orgs] = laws.scope & allowedFlows[context]
}
```

supply chain-wide flows the interaction should enable between companies. To represent a meaningful scope of interaction, functions can be used which use the current context as input. An example is the property-based selection function on line 28, which returns all companies having a given property p and that are visible from the given viewpoint vp.

Roles and Organizations. Roles and organizations are defined on lines 31 and 35. Each role has a field *company*, mapping the role to the company whose agents are allowed to play the role, and a field *capabilities*, representing the capabilities of the role in terms of supply chain flows allowed toward other roles in the organization. Organizations contain the field *roles* representing the current roles of the organization. We also define a help function *enabledFlows* which returns the flows between companies that are enabled by a given organization.

Middleware Model. The state of the middleware is represented by the signature *MiddlewareModel* on line 42. This state is defined as the current context and interaction laws, and the organizations offered by the middleware. A signature fact on line 47 uses the two help functions, we defined earlier, to specify the relation between the organizations offered by the middleware and the current context and interaction laws. The fact specifies that organizations offered by the middleware should enable those, and only those, supply chain flows between companies that are both defined by the scope of the interaction laws and allowed within the current context and its flow policies.

4.2 Asserting Properties

Using the Alloy Analyzer, we can check a number of useful properties of our model. We focus on two relevant properties: (1) asserting that the middleware only offers organizations compliant with the current context; (2) asserting that companies *can* put forward a number of confidentiality constraints, by restricting the supply chain flows in the outsourcing hierarchy. The Alloy specification of these properties is shown in Spec. 2[7]. Both properties have been checked by the Alloy analyzer within a scope of 6 atoms for each type. Although this scope is limited, it covers more than all the possibilities in our running example.

The first property states that companies always need some direct or indirect contractual link, known to the middleware, before their agents can participate in any supply chain-wide interaction. The second property states that a company (*com3*) can restrict all supply chain-wide interactions between any two companies (*com1* and *com2*) that do not have a direct or indirect contractual link with each other independent from the restricting company (*com3*). This property ensures, for example, that 3PLs, such as company 2 in Fig. 4, can restrict the information their subcontractors can expose, such as company 4 and 5. In the example, company 2 allows company 4 to expose information in supply chain-wide interactions, but restricts this for company 5. As a result, the agents

[7] contractPath[com1,com2,context] returns true if a path from com1 to com2 exists in the contractual structure of the given context. indepContractPath[com1,com2,com3,context] returns true if a path exists independent from com3.

of company 1 can participate in an interaction with the agents of company 4, expecting a delay, but not with the agents of company 5, also expecting a delay.

Specification 2 Properties

```
check property1{
  all mw:MiddlewareModel, disj com1,com2:Company |
    !contractPath[com1,com2,mw.context] implies
    no role1,role2:mw.orgs.roles | role1.company = com1 and
    role2.company = com2 and role2 in role1.capabilities.univ
} for 6
check property2{
  all mw:MiddlewareModel, disj com1,com2,com3:Company |
    !indepContractPath[com1,com2,com3,mw.context] and
    (all c1,c2:(client+provider).com3 |
      no Flow->c1->c2 & mw.context.flowPolicies) implies
    no r1,r2:mw.orgs.roles | some r2->Flow & r1.capabilities
      and r1.company = com1 and r2.company = com2
} for 6
```

5 Related Work

The approach presented in this paper intersects with several domains of related work. We focus on a number of representative approaches for business to business (B2B) integration in supply chain management, organization middleware and formal methods for organizations in multi-agent systems.

B2B Integration in Supply Chain Management. Preist et al. [9] recognize the problems of setting up interactions between agents of different supply chain partners, and propose a Web service architecture providing automated B2B integration. Stefansson [11] stresses the importance of automated information sharing in supply chains, but also states the lack of scientific research covering the management of information flows within supply chains. Projects, such as CrossFlow [3], have explored the integration of business process between outsourcing partners using cross-organizational workflow management and virtual organizations. In contrast to the work presented in this paper, these approaches typically focus on the local integration of business processes, lacking explicit support for setting up and managing supply chain-wide interactions.

Organization Middleware. A number of approaches propose middleware-supported organizations and interactions, such as AMELI [2], S-moise+ and ORA4MAS [5], and Law-Governed Interactions [8]. However, most other approaches take an agent-centric perspective in which agents are responsible for performing the functions in organization and managing life cycle of organizations. Novelty toward e-institutions and norm-based approaches is two-folded: (1) Flow policies can specify *local restrictions* on agent interactions. E-institutions

and norm-based approaches typically use global norms rather than company-specific and context-aware restrictions. (2) Implementations of norm-based approaches often rely on central entities enforcing norms, e.g. managers in AMELI and S-Moise+. Our model could also support decentralized realizations [14].

Formal Methods for Organizations. Formalization is recognized as a foundation for analyzing properties such as structure and stability of organizations [1, 12]. Most approaches focus on theoretical aspects of organizations, relying on heavyweight formal methods. Grossi et al. [4], for example, represent organizations as multi-graphs. By adding formal semantics to the graphs, different organizational structures can be compared in terms of performance, flexibility and efficiency. In this paper, we presented a model in Alloy and focused on the management of organizations and domain specific concerns, such as confidentiality. Because Alloy is limited, both in terms of expressiveness and the ability to analyze complex models, alternative approaches such as temporal logic and Petri nets may be more appropriate to explore run-time issues of organizations or complex interaction protocols.

6 Conclusions and Future Work

We have made the case for using an organization middleware to support supply chain-wide interactions in the domain of supply chain management. The organization middleware realizes a collaboration platform and offers organization and role as reusable abstractions to enhance the integration of different business processes. Although we applied our approach to a specific case in logistics management, we have shown how a limited set of organizational abstractions and a light-weight formal modeling language can be used to offer formal guarantees in terms of confidentiality constraints, such as the ability of companies to restrict the interactions between their subcontractors. These guarantees can contribute in establishing the trust of companies in such a middleware approach.

The organizational abstractions, used by the middleware, have proved powerful enough to structure supply chain-wide interactions at a high-level, and enable the separation of managing the interactions and their dynamics from providing the actual functionality provided in the interactions itself. But most importantly, they allow to accurately restrict the interactions among agents, according to company-specific confidentiality constraints.

A prototype implementation of the middleware is also available on the web[8], showing a visual representation of the approach within a controlled setting. Using a web-based GUI, users are able to set up a number of supply chain-wide interactions and dynamically alter the context, flow policies and laws.

Future work. A number of concerns are not addressed by our current model such as dealing with incomplete and incorrect information, security and authentication, and explicit support for interaction protocols, such as automated auctions. Other interesting future directions include a domain specific policy language and integrating the model into a development process.

[8] `http://www.cs.kuleuven.be/~robrecht/AOSE2010/`

Acknowledgement

This research is supported by the Foundation for Scientific Research in Flanders (FWO-Vlaanderen), the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and the Research Fund K.U.Leuven.

References

1. V. Dignum. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. Information Science Reference, 2009.
2. M. Esteva, B. Rosell, J. Rodriguez-Aguilar, and J. Arcos. Ameli: An Agent-Based Middleware for Electronic Institutions. In *AAMAS'04*, pages 236–243. IEEE Computer Society Washington, DC, USA, 2004.
3. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *Computer Systems Science and Engineering*, 15(5):277–290, 2000.
4. D. Grossi, F. Dignum, V. Dignum, M. Dastani, and L. Royakkers. Structural aspects of the evaluation of agent organizations. *LNCS*, 4386:3, 2007.
5. J. Hübner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting Multi-Agent Organisations with Organisational Artifacts and Agents. *Autonomous Agents and Multi-Agent Systems*, pages 1–32.
6. D. Jackson. *Software Abstractions: logic, language, and analysis*. The MIT Press, 2006.
7. N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.
8. N. Minsky and V. Ungureanu. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. *ACM TOSEM*, 9(3), 2000.
9. C. Preist, J. Esplugas-Cuadrado, S. Battle, S. Grimm, and S. Williams. Automated business-to-business integration of a logistics supply chain using semantic web services technology. *LNCS*, 3729:987, 2005.
10. H. Stadtler. Supply chain management and advanced planning: basics, overview and challenges. *European Journal of Operational Research*, 163(3):575–588, 2005.
11. G. Stefansson. Business-to-business data sharing: A source for integration of supply chains. *International Journal of Production Economics*, 75(1-2):135–146, 2002.
12. E. Van Den Broek, C. Jonker, A. Sharpanskykh, J. Treur, and P. Yolum. Formal modeling and analysis of organizations. *LNCS*, 3913:18, 2006.
13. D. Weyns, R. Haesevoets, and A. Helleboogh. The MACODO Organization Model for Context-Driven Dynamic Agent Organzations. *ACM Transaction on Autonomous and Adaptive Systems*, 2010, `http://www.cs.kuleuven.be/~danny/papers/2010TAAS-model.pdf`.
14. D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, and W. Joosen. The MACODO Middleware for Context-Driven Dynamic Agent Organzations. *ACM Transaction on Autonomous and Adaptive Systems*, 5(1):3:1–3:29, 2010.