UNIVERSITE PARIS DESCARTES

Laboratoire d'Informatique Paris Descartes (LIPADE)

Thèse de Doctorat

Informatique/Intelligence artificielle

Nikolaos SPANOUDAKIS

THE AGENT SYSTEMS ENGINEERING METHODOLOGY (ASEME)

Directeur de thèse : Professeur Pavlos MORAITIS

Soutenue le 9 Octobre 2009

Jury :

Massimo COSSENTINO, Chercheur, HDR, CNR-Italy (rapporteur)

Yves DEMAZEAU, DR, CNRS-Grenoble (examinateur)

Amal EL FALLAH-SEGHROUCHNI, Professeur, UPMC (examinateur)

Pavlos MORAITIS, Professeur, Université Paris Descartes (directeur)

John MYLOPOULOS, Professeur, University of Toronto (rapporteur)

ACKNOWLEDGEMENTS

Foremost, I would like to express my gratitude to my supervisor, Pavlos Moraitis for encouraging me to choose this topic and begin my thesis in the city of Paris. I am grateful for his guidance and patience. Pavlos stood not only as supervisor, but also as a friend.

I want to thank Professor John Mylopoulos and Dr Massimo Cossentino for having honored me by expressing their interest in my work and by accepting to be reviewers of this thesis. Also, Professor Amal El Fallah-Seghrouchni and Dr Yves Demazeau for having accepted to be members of my jury and for having honored me with their presence.

Moreover, I would like to thank the staff of Paris Descartes University and the fellow researchers in the lab. We had a great time in the lab working, having coffee breaks, and, later, a beer or two in the city of light. I want to thank the university and the laboratory (LIPADE) administration for funding my participation in numerous conferences and workshops. I want to thank Bruno Bouzy (Maître de Conférences in LIPADE) for giving me the chance to teach the pre-graduate students of the Paris Descartes University on one of my favorite subjects, that of software engineering. I would also like to thank Associate Professor Tom Kontogiannis (from the Department of Production Engineering and Management of the Technical University of Crete) for providing me with software licenses for several CASE tools.

I offer my gratitude to my family, especially my father and mother. They were the ones that made it possible for me to come this far through long years of caring for me with love and good advice.

I would like to dedicate my thesis to my wife, Archontia Aligizaki, who had endless reserves of patience and showed her selfless love by supporting me when I was far away for long periods of time.

Finally, I want to thank all those that I relied on to share my joy and misery, anxiety and hopes, successes and hard times, throughout these four years in France and Greece. These include members of my family, friends in Paris and at home and the colleagues in LIPADE. Thank you guys!

To Archontia

ABSTRACT

This thesis presents on one hand the Agent Modeling Language (AMOLA) for modeling multi-agent systems and on the other hand the Agent Systems Engineering Methodology (ASEME) for developing multi-agent systems. AMOLA provides the syntax and semantics for creating models of multi-agent systems covering the analysis and design phases of a software development process. It supports a modular agent design approach and introduces the concepts of intra-and inter-agent control. The first defines the agent's behavior by coordinating the different modules that implement his capabilities, while the latter defines the protocols that govern the coordination of the society of the agents. The analysis phase builds on the concepts of capability and functionality. AMOLA deals with both the individual and societal aspect of the agents showing how protocols and capabilities can be integrated in agents design. This is the first originality of this thesis, the fact that the inter-agent control model is defined using the same formalism with the intra-agent control model thus allowing the integration of inter-agent protocols in the agent's model as capabilities. ASEME applies a model driven engineering approach to multi-agent systems development, thus the models of a previous development phase are transformed to models of the next phase. This is the second originality of this thesis, the fact that different models are created for each development phase and the transition of one phase to another is assisted by automatic model transformation including model to model (M2M), text to model (T2M) and model to text (M2T) transformations leading from requirements to computer programs. The development process is described using the Software Process Engineering Metamodel (SPEM), the language that is proposed by FIPA for such processes specification. The ASEME Platform Independent Model (PIM) that is the output of the design phase is a statechart that can be instantiated in a number of platforms using existing CASE tools and to an agent platform, the Java Agent Development Framework (JADE). The ASEME and AMOLA presentation is accompanied by two real-world systems presentations (as case studies) that were engineered using this methodology with successful evaluation results and also incorporates a meetings management system development example that demonstrates the whole development lifecycle (from requirements to computer code).

RESUME

Cette thèse présente d'une part, le langage de modélisation d'agents (AMOLA) pour la modélisation de systèmes multi-agents et, d'autre part, la méthodologie pour l'ingénierie des systèmes d'agents (ASEME) pour le développement de systèmes multi-agents. AMOLA fournit la syntaxe et la sémantique pour la création de modèles de systèmes multi-agents couvrant les phases d'analyse et de conception d'un processus de développement logiciel. Il défend une approche de conception modulaire de l'agent et introduit les notions de contrôle intra-et inter-agent. Le premier définit le comportement de l'agent via la coordination des différents modules qui implémentent ses capacités, tandis que le deuxième définit les protocoles qui régissent la coordination de la société des agents. La phase d'analyse s'appuie sur les notions de capacité et de fonctionnalité. AMOLA traite à la fois l'aspect individuel et social des agents en montrant comment les protocoles et les capacités peuvent être intégrés dans la conception d'agents. Une première originalité de cette thèse, est le fait que le modèle du contrôle inter-agents est défini en utilisant le même formalisme que pour le modèle de contrôle intra-agent permettant ainsi l'intégration des protocoles inter-agents dans les capacités de l'agent. ASEME applique un modèle d'ingénierie dirigée par les modèles pour le développement de systèmes multi-agents. Ainsi les modèles d'une phase précédente de développement sont transformées en modèles de la phase suivante. La deuxième originalité de cette thèse est alors le fait que des modèles différents sont créés pour chaque phase de développement et que la transition d'une phase à l'autre est assistée par un dispositif de transformation automatique de modèles, incluant des transformations modèle à modèle (M2M), texte à modèle (T2M) et modèle à texte (M2T) et qui conduisent de la phase des besoins à celle de la programmation. Le processus de développement est décrit à l'aide du Software Process Engineering Metamodel (SPEM), le langage qui est proposé par FIPA pour de tels processus de spécification. Le modèle indépendant de la plate-forme (PIM) d'ASEME, qui est la sortie de la phase de conception est un diagramme états-transitions qui peut être instancié dans un certain nombre de plates-formes en utilisant des outils CASE et aussi dans une plate-forme orientée agent, JADE. La présentation d'ASEME et d'AMOLA est accompagnée de deux présentations de systèmes du monde réel (en tant qu'études de cas) qui ont été conçus en utilisant cette méthode avec succès au niveau d'évaluation de résultats, et également d'un exemple de développement d'un système de gestion de réunions qui démontre toute l'évolution du cycle de vie (depuis les besoins jusqu'au codage informatique).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

Chapter 1

Introduction

Agent oriented development emerges as the modern way to create software. Its main advantage – as referred to by the literature – is to enable intelligent, social and autonomous software development. These three qualities are argued to be the difference with the classic object-oriented design paradigm.

However, one should not focus only on software needs as they are identified by analysts. If the aim is to create modern software with these capabilities one must also look into the software developer's needs. These change in two directions. One is the need to produce more software (in lines of code) because such systems are more complex than traditional systems. Programming depends in writing code in a high level human-understandable language and then translate the code to machine readable format (compile or interpret the program). Throughout the history of computing programming languages gained more abstract semantics always reducing the amount of code that needs to be written for the same program. However, as programs become more complex they tend to need more code. Then, new abstractions emerge lowering the amount of code. Another dimension of this direction is the (semi)automation of code generation so that programs can write big portions of the new programs.

The second is the need to include and integrate diverse, and advanced technologies such as computer vision, planning, argumentative reasoning, etc, that create the need for large software developer teams including many different skills. These needs can only be addressed by a methodology that provides many abstraction levels so that complex software can be modeled. In addition, this methodology must indicate the necessary technologies from the early design stages and allow for clear development tasks decomposition. Moreover, a modular design approach will lead

to a clear and quick composition process. Finally, there is a need for new metaphors that will encapsulate the new qualities of agent technology.

Automation is also an important aspect of modern software development. Model driven engineering is a new paradigm for developing software proposing a software development process that is based on model transformations between the different development phases. A modern methodology should take this trend into account as it caters for non-functional requirements such as portability, interoperability and reusability.

The Agent Systems Engineering Methodology (ASEME) is a methodology for developing multi-agent systems. It started as the Gaia2JADE process for implementing Gaia models (Wooldridge et al., 2000) using the JADE agent platform (Belifemine et al., 2001). It emerged as an evolution of the Gaia2JADE process influenced by the requirements analysis phase of Tropos (Bresciani et al., 2004) and the work of Moore on conversation policies (Moore, 2000). It also reflects the author's experience in engineering real world systems (see, e.g., Matsatsinis et al., 2003, Moraitis et al., 2003b, Moraitis et al., 2005, Moraitis and Spanoudakis, 2007, Spanoudakis and Moraitis, 2009).

The paragraphs present the thesis goals, a small reference to the different steps that led to this thesis, and an outline of this document.

1.1 Thesis Goals

The goal of this thesis is to present the Agent Systems Engineering Methodology (ASEME), which is a methodology for developing multi-agent systems. Its major advantages to existing methodologies are that it builds on existing languages such as statecharts (Harel and Naamad, 1996) and UML, which are familiar to engineers, in order to represent system analysis and design models.

It provides three different levels of abstraction, thus catering for large-scale systems development involving diverse technologies. It is agent architecture and agent mental model independent, allowing the designer to select the architecture type and the mental attributes of the agent that he prefers (e.g. procedural agents, belief-desire-intentions (BDI) agents, etc).

Moreover, the ASEME process follows the modern model driven engineering style, thus the models of each phase are produced by applying transformation rules to the models of the previous phase. Each phase adds more detail and becomes more formal leading gradually to implementation. Thus, ASEME is a model-driven engineering (MDE) process that can be automated by using rules for models transformation and knowledge for adding detail in every development phase.

A platform independent model is the output of the design phase that describes the system and allows its implementation with the use of different platforms or

programming languages. The model transformation process for implementing a multi-agent system using the popular Java Agent Development Environment (JADE) is presented herein. The process is formally presented using the Object Management Group's (OMG) Software Process Engineering Metamodel (SPEM) that has been used in the past for modeling such processes and which is also used by the Foundation of Intelligent Physical Agents (FIPA) agent technology standardization body.

1.2 The Thesis Progression

This thesis started having as a starting point previous work on modeling MAS using the Gaia methodology and implementing them using the JADE framework (Moraitis et al., 2003a), such as the Image system (Moraitis et al., 2003b). Moreover, it was an excellent opportunity to express a point of view on modular agent architectures (Moraitis, 1994; Karacapilidis and Moraitis, 2001; Moraitis, 2002) supported for several years now but not yet matured to a methodology.

As a first activity of this thesis, the Gaia2JADE process was developed (Moraitis and Spanoudakis, 2006) describing the process for combining Gaia and JADE using SPEM. This process was followed for engineering the real world system Im@gine-IT (Moraitis et al., 2005) that was much more complex compared to that of Image as hundreds of personal assistant agents requested services from a network of geographically distributed brokers. Through this work, the limitations of the Gaia2JADE process started to become evident.

In the meantime, research in AOSE showed that a lot of issues were still open (e.g. in Henderson-Sellers and Giorgini, 2005; Dam and Winikoff, 2004). Moreover, the model-driven engineering community matured and provided methods and tools allowing for model transformation, the same for the service oriented engineering community. The work presented in Spanoudakis and Moraitis, 2007a, showed how to integrate a service oriented architecture framework (OSGi and knopflerfish) with an agent platform.

Thus, it emerged ASEME and AMOLA (Spanoudakis and Moraitis, 2007b, 2008a, 2008b). The ASK-IT project was used as a testbed for ASEME. ASK-IT was to a large real world system where hundreds of personal assistant agents requested services from a network of geographically distributed brokers, who in turn consulted a group of specialized assistant agents for mobility impaired persons. The latter deliberated over the needed service for the end user using argumentative reasoning. ASK-IT allowed for experimentation, for example for designing complex protocols there was an effort to use AUML (see Spanoudakis and Moraitis, 2006a), which although could cover the messages exchange part, left the agent program development open, i.e. it had to be defined ad hoc.

ASEME was applied successfully for developing another real-world system, Market-miner (see Spanoudakis and Moraitis, 2008c, 2009). The developed software was

evaluated and succeeded in becoming a candidate for commercialization by a leading Greek software house.

The last part of this thesis was to implement the transformation programs in order to automate the model transformations that had been defined in a theoretical way. This was one of the hardest parts as it entailed the understanding and use of diverse and new technologies (some still in their incubation state, i.e. still not in version 1.0) as three types of transformation were used (i.e. model to model, text to model and model to text).

1.3 Document Outline

The main contribution of this thesis is the presentation of the ASEME methodology and process showing the development steps and their products, as well as the models transformation between the different development phases. The latter allows for traceability of requirements to implementation level and facilitates iteration between the different software development phases. The models that are used by ASEME are defined by the Agent Modeling Language (AMOLA). This thesis contains a working example, the development of a meetings management system, which allows for the understanding of the ASEME process and the AMOLA models. Moreover, the reader will get a wider view of ASEME through the presentation of two real world systems included as case studies (the ASK-IT and MARKET-MINER project results).

Chapter 2 discusses the state of the art in AOSE. It starts with the software engineering discipline in general in order to show the progress of this field and the current trends. Another reason for reviewing software engineering in general is that a lot of works there have influenced the work done within this thesis, mainly the statecharts and UML, but also trends in modular programming, model-driven engineering and agile software development. Then the advances in the AOSE field are presented in two main axes, firstly the existing methodologies, which are presented and discussed, and secondly the approaches to modeling agent interaction protocols as one of the main goals of this work was to create an inter-agent protocol model that would be easily integrated in an agent specification.

Chapter 3 presents the AMOLA models for the requirements analysis, analysis and design phases. Some of the most important results of this thesis are presented in this section, i.e. the formal definition of the liveness formula of a role model and the formal definition of a statechart based on the ordered rooted tree. The different AMOLA models are presented using examples from the ASK-IT real-world system conception.

Following, in Chapter 4, the ASEME process is presented. This chapter starts by providing the reasons why there is still room for a new methodology in AOSE and what are the challenges related to the proposal of one. It shows how and when the models of AMOLA are used in the software development phases and another

important result of this work, how models of a previous phase are transformed to models of a next phase. The ASEME process presentation is facilitated by a working example, that of the meetings management system. At the end of the chapter the reader will find a case study for developing a real-world agent-based system, MARKET-MINER. This case study demonstrates how to analyze and design an agent-based system using the ASEME process. It also shows how a logic-based reasoning mechanism was integrated in an AMOLA design and how to get an agent prototype using a CASE tool available in the market.

Chapter 5 is concerned with proving the feasibility of the transformations defined in the previous chapter and also with presenting and discussing the enabling technologies for the transformation tasks. These are diverse technologies encompassing the whole model-driven engineering spectrum as the transformation types used include model to mode (M2M), text to model (T2M) and model to text (M2T) transformations. This is another originality of this methodology, the fact that it includes three transformation types. The meetings management system is modeled throughout this chapter showing what information is added at what model and how the models of a previous phase are transformed to those of a next phase. This example starts from the requirements analysis and goes through the development phases up to code generation. All the AMOLA metamodels, transformation programs and generated models are presented in this chapter.

Chapter 6 presents another aspect of the AMOLA design model. Its capability to be transformed to a process model. Unfortunately, the tools that were available for process modeling did not import any kinds of models so the transformation process is manual. However, the capability of such process models to be used for verification and simulation of system properties but also for evaluating the scalability of the systems is demonstrated through a case study done in the context of the ASK-IT project.

Chapter 7 discusses the future perspectives of this work. They are identified in two directions. The first direction is in further evaluating and expanding the ASEME process. This work is about implementing better graphical editors for the AMOLA models, expanding the automatic code generation capabilities and possibilities and further evaluating it through case studies. The second is related to further research directions which are numerous and in very interesting fields (at least for the writer).

The thesis is concluded in Chapter 8, which summarizes its findings and results. A number of annexes include the references, the abbreviations used throughout this thesis and all the details related to the presented case studies, the programs that were written for the ASEME transformation processes, the AMOLA metamodels and the files related to the meeting management sample project.

Chapter 2

State of the Art and Related Work

The state of the art presentation starts with an overview of the evolution of software engineering also covering its modern trends. Then, it focuses on Agent Oriented Software Engineering (AOSE) firstly by discussing how it emerged as a scientific field and then by presenting in detail and discussing the achievements so far.

2.1 Software Engineering

According to the IEEE Computer Society, *software engineering* is defined as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software and the study of such approaches (IEEE, 1990). In the writer's point of view, software engineering emerged as soon as computer programs (or information systems) became products that would be used by people other than those who built them. Thus, on one hand the development process had to be explained and allocated a budget in a rational way and on the other hand different engineers should be able to be involved in the process, thus all the steps should be adequately documented.

The software engineering field has tools such as process models and methodologies (or simply methods). The term *process model* guides a software project and provides answers to the following questions (Boehn, 1988):

1. *"What shall we do next?*
2. *How long shall we continue to do it?"*

The methodologies are concerned with different issues such as the products outputted by each phase and how to navigate through each phase. Tolvanen (1998) provides the following definition for a software method:

"A predefined and organized collection of techniques and a set of rules which state by whom and in what order the techniques are used."

2.1.1 Structured Programming

In the beginning, when procedural languages were used for programming, software engineering focused on defining data flows and processes that handled the data. The Structured Systems Analysis and Design Method (SSADM) is a representative of that era and its dominant models were the Data Flow Diagrams (DFD) as they were proposed by Stevens et al. in 1974.

2.1.1.1 Modeling Methods

Data Flow Diagrams

DFDs define the software processes and the data structures or external systems that they use (either to read or write information). These concepts are graphically displayed using the notations provided in Figure 1. The modeler uses different levels of abstraction that allow the whole system under development to be represented as a process accessing and modifying numerous data sources or external systems. As the modeler adds detail in subsequent views the original process is replaced by many more specialized ones.

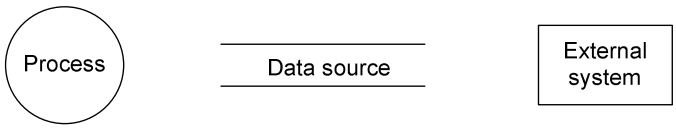


Figure 1. The Data Flow Diagrams notation

The reader can get an idea about DFDs by observing how a security software system is modeled. In Figure 2 the top level of the system is displayed. In this level the whole system is viewed as one process. This system gets information from a control panel and several sensors and outputs information to the control panel display. It can also output information to an alarm and through the Public Service Telephone Network (PSTN) to the police. Figure 3 zooms in the next level (Level 1.1) where more detail is added (more specialized processes and clearer data-flow) to the single process of level 1.

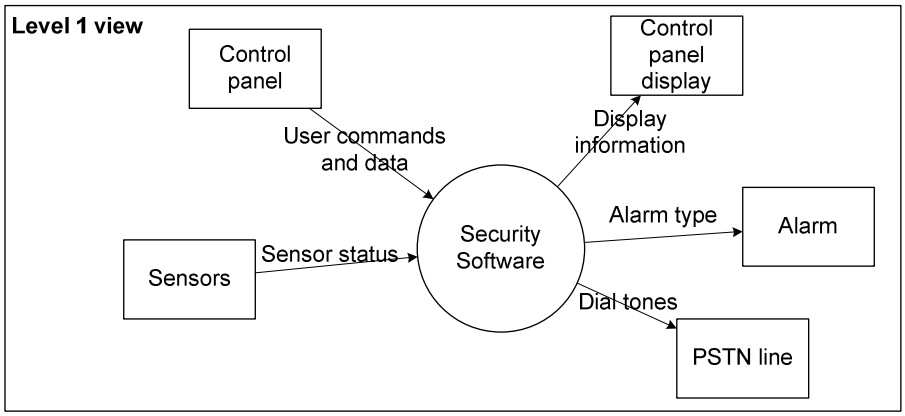


Figure 2. The Security Software level 1 DFD.

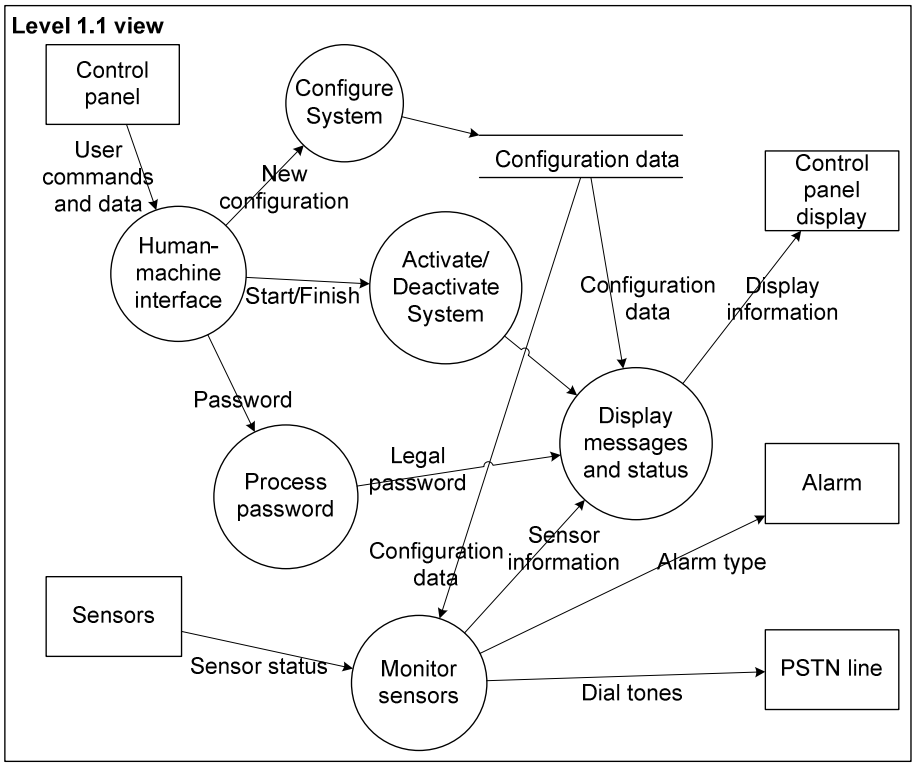


Figure 3. The Security Software level 1.1 DFD.

The Z language

Later, in the late 80s more formal methods that supported a graphical notation, like the Z language (Spivey, 1989), started to emerge. Their goal was to model systems and be capable to validate them before implementation. Z used elements from set theory and logic and allowed the use of the same formalism for modeling data structures and functions (see the Z language notation in Figure 4). For example, in Figure 5, the reader can inspect a sample model including the *Tank* data structure along with the *Fill Tank* function. According to the figure, the *Tank* has a *Container*

and a *Sensor*. The *Container* has a *Capacity* of *100* units and the *Reading* of the *Sensor* is the *Content* of the *Tank*. The *Fill Tank* function is used for adding a *Quantity* in the *Tank* except in the case that the outcome would exceed its capacity in which case a *Message* is outputted and no action is taken.

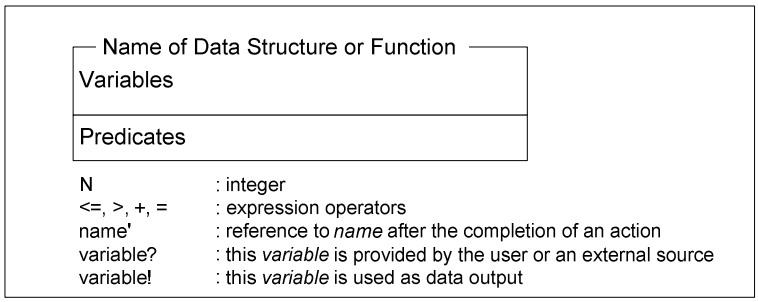


Figure 4. The Z language notation

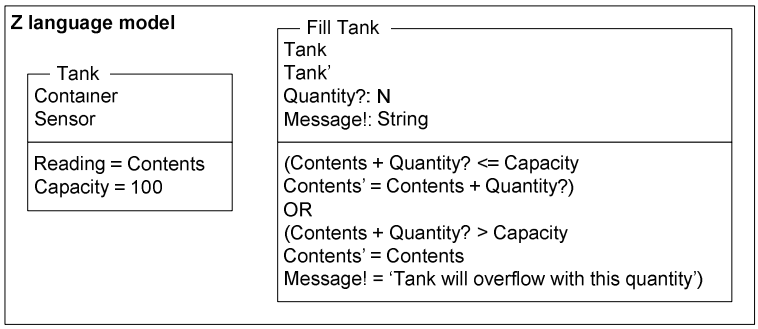


Figure 5. Using the Z language for modeling data and functionality.

2.1.1.2 Software Processes

The waterfall model

SSADM relied on a waterfall development model (Royce, 1970) which defined clearly distinguished successive development phases with the possibility of iteration. Those phases were the:

1. Requirements analysis. In this phase the system requirements are gathered and documented.

2. Analysis. In this phase the requirements are transformed to technical needs for the hardware and software that must be included in the system.

3. Design. In this phase the system is modeled using software engineering methods

4. Implementation. In this phase the system is developed according to the plans of the previous phase.

5. Verification-Validation. This phase is mostly concerned with system performance and correctness testing.

6. Maintenance. The software is considered a living system that needs to be maintained until the end of its lifecycle. Maintenance is about correcting arising problems after system delivery, adding or extending the system functionality.

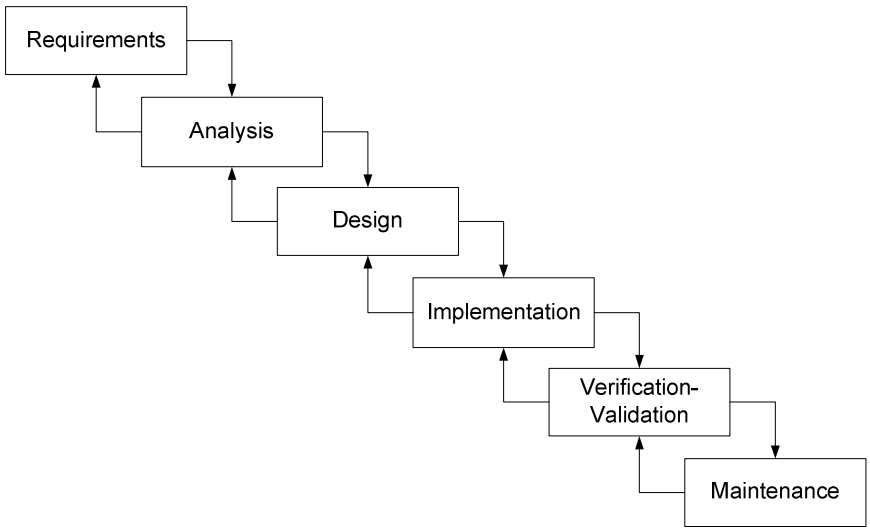


Figure 6. The waterfall development model.

2.1.2 Object Oriented Development

When the object-oriented engineering paradigm emerged, new concepts were used such as classes, objects, polymorphism, inheritance, etc. According to Young (1992) Object oriented programming is a new metaphor to the way a system is designed. It is a programming technique that gives emphasis to the objects of a system instead of the tasks that the system must undertake.

Object-oriented design (OOD) made its appearance in 1982 in a paper written by Booch (1982). After that date, many researchers proposed new ways for modeling systems, incorporating the object oriented programming (OOP) new concepts in their models, and, finally, the most important technology that emerged was the Unified Modeling Language (UML, 2005), whose first version appeared in 1997 by the Object Management Group (OMG). In structured programming, tasks were refined in a top-down approach so that in the end small functions could be assigned to the developers for coding, while in object-oriented design the system functionality

is provided by a number of interacting objects. The latter can be assigned to developers for coding.

The processes that emerged with the object-oriented programming paradigm introduced the concept of iteration, i.e. the fact that a software system is developed gradually through development cycles during each one of them the different development phases' products become more detailed and resemble more closely the desired outcome.

2.1.2.1 Modeling Methods

UML

The prevailing method in OOD is UML. UML defines the class diagrams for modeling the concepts of class and inheritance. Classes define both the data and the functions that use or create them. The inheritance concept is depicted in Figure 7. Classes of objects are defined grouping all object properties. Then, more specialized objects are derived from each class adding detail. For example, the *Jet* is a special case of a *Flying Vehicle*, which in turn is a special case of a *Vehicle*. The final *Jet* class shown in grey background includes all the attributes defined in its predecessors.

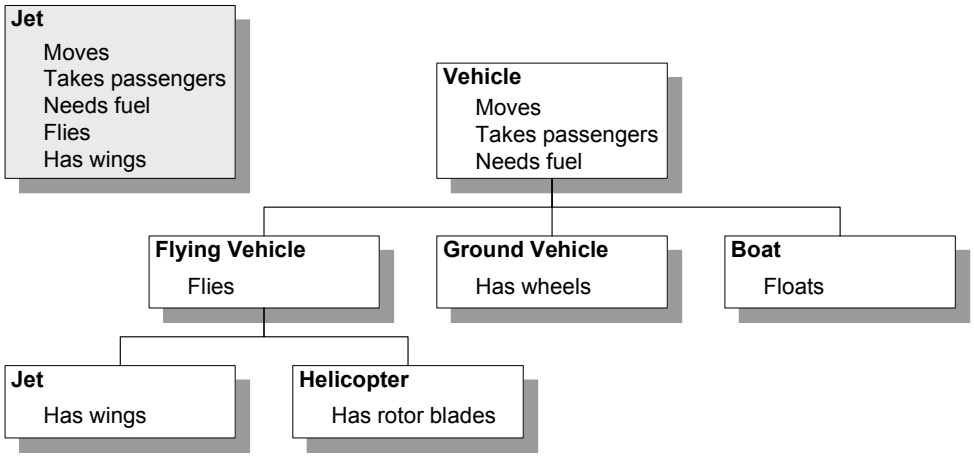


Figure 7. Classes and inheritance

The attributes of a class can be defined as private, protected or public depending on the level of access that other objects will have to the objects of the class. The objects of a class are also called *instances* and they can be different based on the values of their attributes.

A class can also define methods that provide functionality to the object that invokes them. The objects can invoke other objects' methods through message passing. *Polymorphism* allows different descendant classes of one class to respond to the same message with their individual way. Thus (referring to the example of Figure 7) a

Vehicle can receive a message to move but this method can be implemented in a different way by a *Helicopter* and a *Jet*.

A *class diagram* can be used for modeling classes and for defining relationships between the classes. Statecharts (Harel and Naamad, 1996) can be used for defining a class behavior when it depends on the sequence by which its methods are invoked. Other types of diagrams are also used by UML such as *sequence diagrams* (for defining scenarios of messages exchanging between objects) and *activity diagrams* (showing workflows that can involve one or more objects) - in many ways UML activity diagrams are the object-oriented equivalent of data flow diagrams (Ambler, 2004).

UML also defines models for the analysis phase. Such are the *use case diagrams*, which model the functionality provided by the system showing the involved actors, their goals represented as use cases, and any dependencies among those use cases, using the *include* or *extend* association types. The first suggests that a use case includes the functionality of the included one. The second suggests that a use case extends (somehow modifies) the functionality of another use case.

2.1.2.2 Software Processes

The Spiral model

In the late 80s, the development process connected the last phase of the waterfall model to the first and embraced new ideas such as prototyping and simulation denoting that software systems were to be developed gradually. Thus, the spiral development model emerged (Boehm, 1988). The spiral model, depicted in Figure 8, proposes software development in successive iterations of four phases. After each iteration, more detail has been added to the system under development, thus coming closer to the final result.

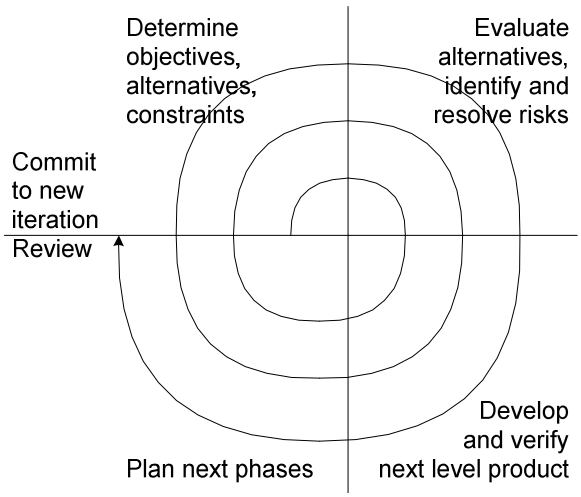


Figure 8. The spiral model.

A typical cycle of the spiral includes four steps (phases):

1. Identify the objectives related to the next implementation phase (e.g. increase performance, add functionality, etc), the alternative means of implementation (e.g. competing technologies) and the constraints (e.g. in cost)
2. Evaluate the alternatives relative to the objectives and constraints and compute the risk related to each one of them
3. Choose, develop and test the best alternative
4. Evaluate the outcome of the previous phase and plan the next cycle of development

At the end of each cycle the progress of the project is reviewed and the decision makers decide whether they should continue supporting this project or not (in the case that this is not the last iteration). If they decide to continue a new cycle begins with new goals and constraints.

The spiral model can accommodate most of the previously proposed development models as special cases. For example, in the case of system development using only one, carefully planned, iteration the spiral model can resemble the waterfall model.

The Rational Unified Process

The Rational Unified Process (Kruchten, 2000) is a software development process using UML. It is iterative and its phases can include more than one iterations. In Figure 9 the reader can see the different phases of RUP (in the horizontal axis that also functions as the time axis) and the amount of work required in the different disciplines that are related to a software development project (shown on the vertical axis). The surface of the bar related to each discipline defines the amount of work needed and at the points of time where the bar is higher that is when most of the resources related with the discipline are spent.

RUP defines a set of artifacts, activities and roles related to each discipline and to each phase. The four phases have the following goals (Hirsch, 2002):

1. *Inception*: Define the project objectives
2. *Elaboration*: Define system architecture and plan the next phases
3. *Construction*: System implementation
4. *Transition*: Beta-test and release the system

Like in the spiral model each iteration ends with a version of the system. The results of the iteration are assessed and the goals for the next one are set.

The new concepts used in RUP with relation to previous processes is the identification of business modeling, which is about describing the business processes and the internal structure of a business in order to better understand it and better

define the software requirements. The environment discipline is about adapting RUP to the needs of a specific project.

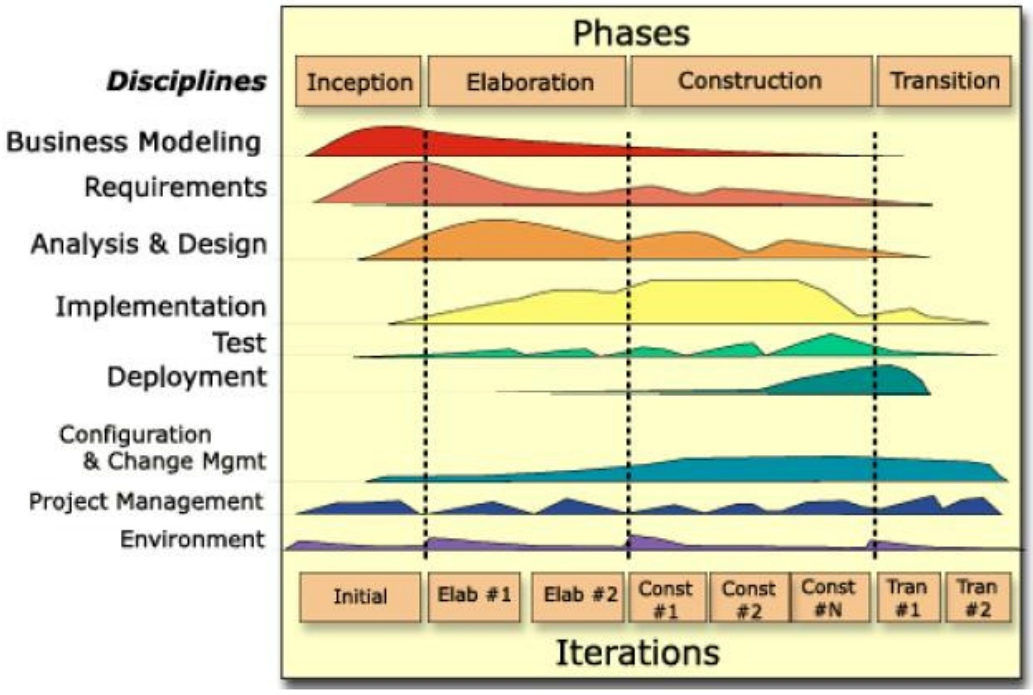


Figure 9. The Rational Unified Process (Hirsch, 2002).

2.1.3 Statecharts

Statecharts (Harel and Naamad, 1996) are used for modeling systems. They are based on an activity-chart that is a hierarchical data-flow diagram, where the functional capabilities of the system are captured by activities and the data elements and signals that can flow between them. The behavioral aspects of these activities (what activity, when and under what conditions it will be active) are specified in statecharts.

There are three types of states in a statechart, i.e. OR-states, AND-states, and basic states. OR-states have substates that are related to each other by “exclusive-or”, and AND-states have orthogonal components that are related by “and” (execute in parallel). Basic states are those at the bottom of the state hierarchy, i.e., those that have no substates. The state at the highest level, i.e., the one with no parent state, is called the root. The state hierarchy and the different types of states are demonstrated in Figure 10. States S, B, C, D are OR-states, state A is an AND-state and states B1, B2, C1, C2, D1, D2, E are basic states. In this case, the root state has state S as a substate. The *active configuration (AC)* is a maximal set of states that the system can be in simultaneously. Any active configuration includes the root state,

exactly one substate of each OR-state and all substates for each AND-state contained. For example, the sets {root, S, A, B, C, D, B1, C1, D1}, {root, S, A, B, C, D, B2, C2, D1} and {root, S, E} are valid active configurations of the statechart depicted in Figure 10.

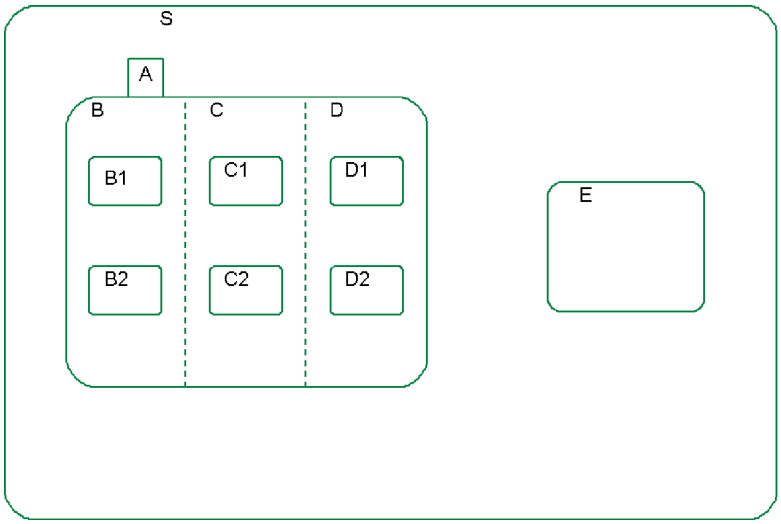


Figure 10. The hierarchy of states in a statechart (Harel and Kugler, 2004).

Each transition from one state (source) to another (target) is labeled by an expression, whose general syntax is e[c]/a, where e is the event that triggers the transition; c is a condition that must be true in order for the transition to be taken when e occurs; and a is an action that takes place when the transition is taken. All elements of the transition expression are optional.

Moreover, there are *compound transitions (CT)*. These transitions can have more than one source or target states. There are two kinds of CTs, *AND-connectors* and *OR connectors*. AND connectors are of two types, *joint transitions* (more than one sources, see Figure 11) and *fork transitions* (more than one targets, see Figure 12). The most commonly used OR-connector is the *condition transition* (see Figure 13).

Figure 14 demonstrates the fact that only full CTs can cause a state transition. If t1, t2 and t3 are ready to execute they form an initial CT. However, this initial CT needs a continuation CT that includes default connectors. Thus, joined by the default connectors t4 and t5 the initial CT becomes the full CT that can be executed {t1, t2, t3, t4, t5}, since a transition must lead to a valid active configuration.

The *scope* of a transition is the lowest level OR-state that is a common ancestor of both the source and target states. When a transition occurs all states in its scope are exited and the target states are entered.

Multiple concurrently active statecharts are considered to be orthogonal components at the highest level of a single statechart. If one of the statecharts

becomes non-active (e.g. when the activity it controls is stopped) the other charts continue to be active and that statechart enters an idle state until it is restarted.

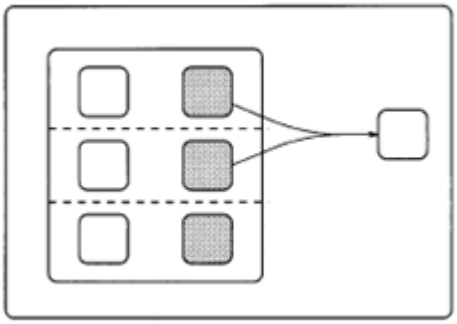


Figure 11. A joint transition. The grey states are those exited when the transition is taken (Harel and Naamad, 1996).

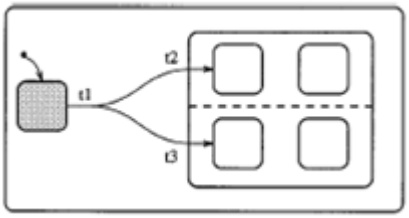


Figure 12. A fork transition. The grey state is the one exited when the transition is taken (Harel and Naamad, 1996). All t1, t2 and t3 must be executed.

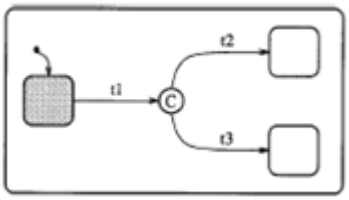


Figure 13. A condition transition. The grey state is the one exited when the transition is taken (Harel and Naamad, 1996). t1 and t2 or t1 and t3 will be executed.

Statecharts were used for modeling solutions using procedural languages (e.g. C) in *STATEMATE* (Harel and Naamad, 1996) and *VisualSTATE* (Wasowski, 2005). In their work, Harel and Kugler (2004) proposed the semantics for modeling object oriented systems using the statecharts language in the *Rhapsody* tool. The main difference with the previous work is in the execution semantics allowing for multi-threading and message passing (synchronous and asynchronous) between objects. They also introduced the possibility to add a special timeout event that could trigger transitions. They define different statecharts for each class to be developed.

However, each instance of the class (i.e. object) can be in a different active configuration in runtime. Each class defines the set of events that it can receive.

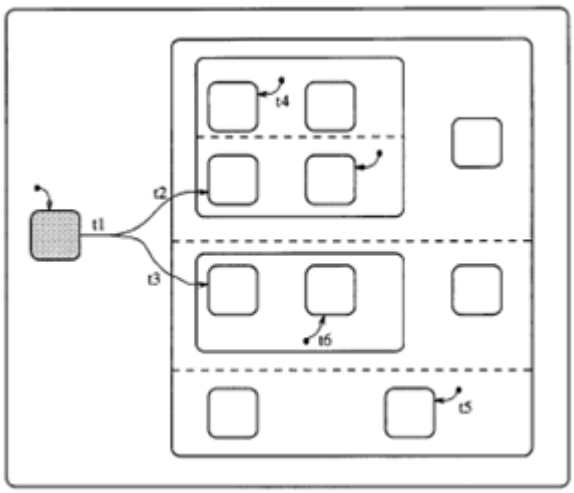


Figure 14. Demonstrate how only full CTs reach a next state (Harel and Naamad, 1996).

2.1.4 Modern Approaches to Software Engineering

2.1.4.1 Agile processes

The latest software engineering techniques are extreme programming and agile processes that emphasize on the facts that the client should be involved in all the software development phases and that huge systems needed huge models that were very costly to develop and maintain in an organization.

The agile development methodologies appeared in the start of the 21st century declaring a manifesto with 13 principles (Fowler and Highsmith, 2001). These principles reflected the modern needs of software development, i.e. the need for addressing continuously changing requirements, continuous evaluation, the need for motivated individuals (who need to exploit new technologies as they appear) and, finally, the need for less bureaucracy related to the extensive production of models that few people (only the developers) can read.

The need for the agile methods has best been described by Boehn (2002):

"Plan-driven methods work best when developers can determine the requirements in advance—including via prototyping—and when the requirements remain relatively stable, with change rates on the order of one percent per month."

Plan-driven methods are those that begin with the solicitation and documentation of a set of requirements that is as complete as possible (Pikkarainen, 2008).

Many different agile approaches such as XP (Beck, 2000), Scrum (Schwaber and Beedle, 2002), Crystal (Cockburn and Highsmith, 2001), and others (see Pikkarainen, 2008, for a complete list) show that agile processes are a real industry trend. Other researchers, such as Hirsch (2002) claim that they can use an existing process, i.e. RUP, for agile development just by narrowing the artifacts usage (in his paper he identifies 10 to 12 needed out of more than 80 RUP artifacts). He also describes successful projects that were developed by four persons while RUP identifies 40 roles participating in the software development process. Hirsch used these roles in order to identify the competencies needed for achieving an activity and use them as a checklist for his personnel in order to assign responsibilities.

2.1.4.2 Modular Programming

In computing, a module is a software entity that groups a set of (typically cohesive) subprograms and data structures. Modularization means that functionality is packaged and divided into small units (Meyer, 1997). Modules promote encapsulation (i.e. information hiding) through a separation between the interface and the implementation. Modules can also be seen as computational elements that other modules can use (Braubach et al., 2005, Ghezzi et al., 2002). Modules hide their internal information and they may change their implementation without affecting other modules. They are treated as black boxes when introduced in an information system. Szyperski (1997) defines the term component:

"A software component is a binary unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

Especially in large, complicated programs, modularity is a desirable property. Even in Procedural Programming, modularity is proposed to be implemented using procedures that have strictly defined channels for input and output. Inputs are usually specified syntactically in the form of arguments and the outputs delivered as return values. Scoping is another technique that helps keep procedures strongly modular. It prevents the procedure from accessing the variables of other procedures (and vice-versa), including previous instances of itself, without explicit authorization. This helps prevent confusion between variables with the same name being used in different places, and prevents procedures from stepping on each other's feet. Because of the ability to specify a simple interface, to be self-contained, and to be reused, procedures are a convenient vehicle for making pieces of code written by different people or different groups.

Sophisticated forms of modularity became possible with object-oriented programming. Instead of dealing with procedures, inputs, and outputs, object-oriented programs pass around objects. Computation is accomplished by asking an object to execute one of its internal procedures (or one it has inherited), possibly drawing on some of its internal state. Indeed, the "module" abstraction is considered as one of the main conceptual advantages of object orientation (Booch, 1994).

2.1.4.3 Service-oriented Architecture (SoA)

This paragraph is not following the previous one without a reason as according to Cervantes and Hall (2004) service orientation uses the idea of assembling a system from modular building blocks, with the difference that these building blocks are *services*. The difference between services and components is that the first are contractually defined in a service description that contains syntactic, semantic and behavioral information. Components, on the other hand need to describe more than that, actually how they would be integrated in a computer program. Thus, the idea of services is that not only they do not need to be integrated physically in a new program or deployed with a new system, but they may have to be searched for and executed on run time, that is there may have been no knowledge about them during the time of system (or new services) development.

Bennett et al. (2000) argue that in the future, software will be delivered as a service within the framework of an open marketplace. In this sense *SoA* can be considered as a marketplace, where a s*ervice* is an individual shop/trader in the market.

2.1.4.4 Model-driven Engineering

MDE (Beydeda et al., 2005) is the systematic use of models as primary engineering artifacts throughout the engineering lifecycle. It is compatible with the recently emerging Model Driven Architecture (MDA) paradigm (see Kleppe et al., 2003). MDA's strong point is that it strives for portability, interoperability and reusability, three non-functional requirements that are deemed as very important for modern systems design. MDA defines three models:

- A computation independent model (CIM) is a view of a system that does not show details of the structure of systems. It uses a vocabulary that is familiar to the practitioners of the domain in question as it is used for system specification.

- A platform independent model (PIM) is a view of a system that on one hand provides a specific technical specification of the system, but on the other hand exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms. The system is described in platform independent format at the end of the design phase.

- A platform specific model (PSM) is a view of a system combining the specifications in the PIM with the details that specify how that system uses a particular type of platform.

Model driven engineering relies heavily in model transformation (Sendall and Kozaczynski, 2003). 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. *Meta* is a prefix originating from the Greek word "μετά" meaning "after", which is used in epistemology to mean "about".

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. Thus, like a computer program conforms to the grammar of the programming language in which it is written a model conforms to its metamodel (or its *reference model*). 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 (Jouault and Bézivin, 2006). We adopt the following three definitions from the same work:

Definition 2.1. A *metametamodel* is a model that is its own reference model (i.e. it conforms to itself).

Definition 2.2. A *metamodel* is a model such that its reference model is a metametamodel.

Definition 2.3. A *terminal model* is a model such that its reference model is a metamodel.

We call these levels: M1, M2 and M3. M1 consists of all models that are not metamodels. M2 consists of all metamodels that are not the metametamodel. M3 consists of a unique metametamodel for each given technical space. Figure 15(A) shows how to adapt the definition of model to this three-level modeling stack. Figure 15(B) shows the associations between the three level models according to the above definitions. Throughout this thesis, the word model will usually refer to a terminal model.

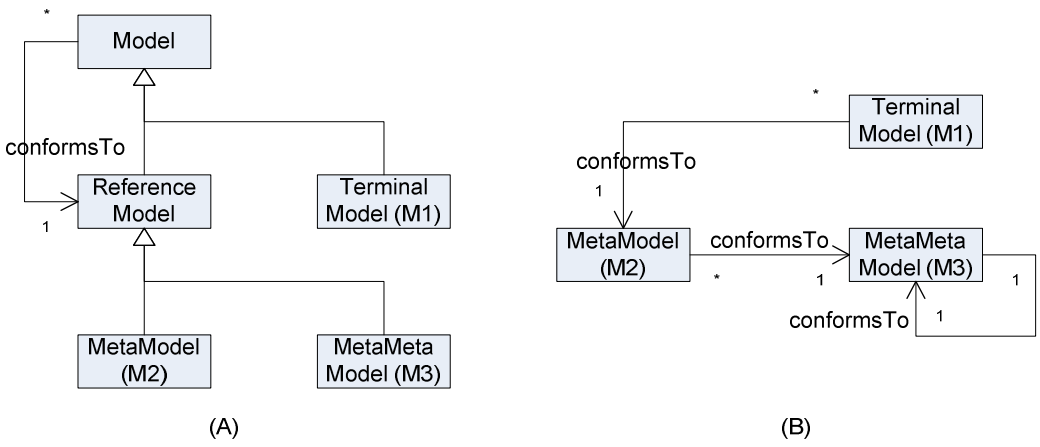


Figure 15. Metamodeling stack representation (A) with model definition (B).

The structure for models defined in this section is compatible with the OMG view as illustrated in the MDA guide (see the Object and Reference Model Subcommittee, 2005). An example of this approach is the EBNF technical space: programs (M1) adhere to grammars (M2), which adhere to the grammar of EBNF (M3).

After having defined the models of models (or metamodels) it is possible to define transformations of a model to another model. The Object Management Group issued a Request For Proposals (RFP) in 2002 titled Query/Views/Transformations (QVT) aiming to define a language for defining model transformations. The collective response for this CFP is referred to as QVT (Object Management Group, 2005). In the same time, Jouault and Kurtev (2006b) proposed the ATLAS transformation language (ATL) for model transformation adhering to the same requirements as QVT.

The overall scheme of the model transformation process followed by both ATL and QVT is presented in Figure 16. On the top there is a common metametamodel (MMM) to which conform two metamodels (MMa and MMb). The goal of the model transformation process or *model to model* process (abbreviated as *M2M*) is to take a model Ma, which conforms to MMa, as input (or *source model*) and produce the Mb, which conforms to MMb as output (or *target model*).

Besides the source and target models the process executes a transformation program (let it be called Tab). The Tab describes the procedure for transforming a model that conforms to MMa to a model that conforms to MMb. The transformation program itself is a model that conforms to a metamodel (MMt), which in turn conforms to the metametamodel (MMM). Thus, like in the case of EBNF, MMt defines the abstract syntax of the transformation language. Both QVT and ATL define their abstract syntaxes through such a metamodel.

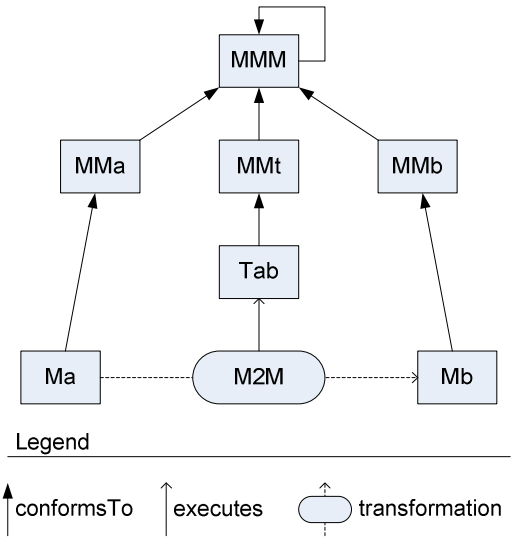


Figure 16. The general scheme of model transformation

2.2 Agent Oriented Software Engineering

Agent Oriented Software Engineering (AOSE) emerged after the autonomous agents and multi-agent systems was established as a research field of the computer science/artificial intelligence discipline. The first workshop with this name took place in 2001 (Ciancarini and Wooldridge, 2001), although the term had already appeared in earlier works, e.g. in Jennings (1999).

Agent-oriented development is viewed as a next step in software engineering evolution. Agents are the descendants of objects. The new ideas incorporated in the agent concept that characterize the notion of agency (Wooldridge and Jennings, 1995; Weiss, 2003) are also their main differences with objects (Odell, 2002):

- *Autonomy*. Agents can operate without the direct intervention of humans or other entities, and can have some kind of control over their actions, internal state and resource consumption

- *Social ability*. Agents use a communication language to interact with other agents (and possibly humans). They have some kind of control over their acquaintances and can choose their collaborators for problem solving

- *Reactivity*. Agents perceive their environment and respond in a timely fashion to changes that occur in it according to their goals

- *Pro-activeness*. Agents are able to exhibit goal-directed behaviour by taking the initiative, be purposeful, and not simply act in response to the environment changes.

Other characteristics of agents are *adaptability* (the agent can adapt to changes in its environment) and *persistence* (an agent has a lengthy persistence, unlike objects that are instantiated to do something and then are sent to the garbage collector).

Agents originated from the *distributed problem solving* or *distributed artificial intelligence* discipline. This discipline argued that it is more efficient to create specialized problem solvers (agents) who can, through interaction, provide solutions to more complex problems than the ones that one of them can solve by itself (O'Hare and Jennings, 1996). Systems that are composed of interacting agents are also termed as multi-agent systems (MAS).

The new characteristics and concepts of multi-agent systems and autonomous agents needed to be integrated in a software engineering methodology. AOSE came to cover this need. Until today, a number of methodologies have been proposed each supporting different styles of agent programming and different agent architectures. Thus, it emerged, the need for combining method fragments from different methodologies. Method fragments are reusable methodological parts that can be used by engineers in order to produce a new design process for a specific situation (see Cossentino et al., 2007, for details). This allows a development team to

come up with a hybrid methodology that will support the needs of specific programming and modeling competencies.

In what follows the most important methodologies in the literature, in the sense that they introduce new ideas and methods for modeling a MAS, are presented. The methodologies are viewed from the perspective of the papers and books that proposed them, but also from the perspective of the writer and other works that compare AOSE methodologies, such as those of Henderson-Sellers and Giorgini (2005) and Dam and Winikoff (2003). Moreover, important works in the area of modeling inter-agent protocols are also presented. One of the major issues in Agent Oriented Software Engineering (AOSE) is the modeling, representation and implementation of agent interaction protocols. A wide range of methodologies for AOSE either adopt one existing model (most usually AUML), while others either employ UML models (like activity diagrams), or do not address the issue and just define messages that the agents send to each other (allowing the modeling of simple protocols).

2.2.1 Multi-agent Systems Engineering (MaSE)

MaSE (Deloach et al., 2001) defines a process for building MAS with two phases, the analysis phase and the design phase. During the analysis phase three activities take place: The *capturing goals* activity is about defining the system goals and also organizing them in a goal hierarchy. The next activity is about *applying use cases* which builds a set of sequence diagrams corresponding to system usage scenarios. The third activity is concerned with *refining roles* by defining the *role model* that describes the roles in the system, their goals and the tasks they need to complete in order to achieve them and communication links between the roles. During this activity each task is defined as a finite state machine in the *concurrent task model*.

In the design phase, the first activity is about *creating agent classes*. In a new type of diagram, the agent class diagram, each agent type is defined as a class whose attributes are the roles that it aggregates. The agent class connects to other classes indicating the possible interactions or *conversations*. The latter are refined in the next activity of this phase, i.e. *constructing conversations*. Towards this end, another type of diagram, i.e. the communication class diagram, which is also in the form of a finite state machine, is employed. The third activity is about assembling the agent classes, a step which aligns the previous models to an implementation platform. Finally, in the forth activity of this phase the system components deployment is decided outputting a relevant diagram.

MaSE is supported by agentTool (DeLoach and Wood, 2000), a tool allowing for the usage of the analysis and design artifacts including an automated transformation process for the analysis models to design models.

All in all, MaSE defines a system goal oriented MAS development methodology. The authors define for the first time inter and intra-agent interactions that must be integrated. However, in their models they fail to provide a modeling technique for

analyzing the systems and allowing for model transformation between the analysis and design phases. Their concurrent tasks model derives from the goal hierarchy tree and from sequence diagrams in a way that cannot be automated. MaSE agents are related to system goals. This restricts the definition of autonomous agents.

O-MaSE (Deloach, 2005) introduced the organization concept in MaSE aiming to overcome MaSE's limitations towards inter-agent protocol modeling and situation of the MAS in the environment, introducing the use of AUML (see §2.2.3) in MaSE.

2.2.2 The Gaia Methodology

The Gaia methodology (Wooldridge et al., 2000; Zambonelli et al. 2003) is an attempt to define a general methodology that it is specifically tailored to the analysis and design of MAS. Gaia emphasizes the need for new abstractions in order to model agent-based systems and supports both the levels of the individual agent structure and the agent society in the MAS development process. Gaia adds the notion of *situatedness* to the agent concept. According to this notion, the agents perform their actions while situated in a particular environment. The latter can be a computational environment (e.g. a website) or a physical one (a room) and the agent can sense and act in the environment.

MAS, according to Gaia, are viewed as being composed of a number of autonomous interactive agents that live in an organized society in which each agent plays one or more specific roles. Gaia defines the structure of a MAS in terms of a role model. The model identifies the roles that agents have to play within the MAS and the interaction protocols between the different roles. The Gaia methodology is a three phase process and at each phase the modeling of the MAS is further refined. These phases are the analysis phase, the architectural design phase and, finally, the detailed design phase.

The objective of the Gaia analysis phase is the identification of the roles and the modeling of interactions between the roles found. Roles consist of four attributes: *responsibilities*, *permissions*, *activities* and *protocols*. Responsibilities are the key attribute related to a role since they determine the functionality. Responsibilities are of two types: *liveness properties* – the role has to add something good to the system, and *safety properties* – the role must prevent something bad from happening to the system. Liveness describes the tasks that an agent must fulfill given certain environmental conditions and safety ensures that an acceptable state of affairs is maintained during the execution cycle. In order to realize responsibilities, a role has a set of permissions. Permissions represent what the role is allowed to do and, in particular, which information resources it is allowed to access. The activities are tasks that an agent performs without interacting with other agents. Finally, protocols are the specific patterns of interaction, e.g. a seller role can support different auction protocols. Gaia has operators and templates for representing roles and their attributes and also it has schemas that can be used for the representation of interactions between the various roles in a system.

The operators that can be used for liveness expressions-formulas along with their interpretations are presented in Table 1. Note that activities are written underlined in liveness formulas.

Table 1. Gaia Operators for Liveness Formulas

Operator	Interpretation
x . y	x followed by y
x | y	x or y occurs
x*	x occurs 0 or more times
x+	x occurs 1 or more times
x^{ω}	x occurs infinitely often
[x]	x is optional
x || y	x and y interleaved

The reader can see in Figure 17 a Gaia roles model for a role named "TravelGuide". This role employs seven protocols and six activities (activities are underlined in the *Protocols and Activities* field). In its liveness formula it describes the order that these protocols and activities will be executed by this role. In Figure 18 the "RequestMap" protocol is presented as a Gaia interactions model. This model shows the interacting roles, in this case a PersonalAssistant (the initiator) and a TravelGuide (the partner role) and the conditions under which it is initiated by the initiating role (on the bottom left side of the figure). On the bottom right side of the figure the outcome of the interaction is described.

Role: TravelGuide (TG)
Description: It wraps a Geographical Information System (GIS). It can query the GIS for routes, from one point to another.
Protocols and Activities: RegisterDF, QueryGIS, InvokeGetRouteGISFunction, InvokeGet-NearbyPOIsGISFunction, InvokeGetMapGISFunction, InvokeGetPOIsInfoGISFunction, RequestRoutes, RespondRoutes, RequestMap, RespondMap, RequestNearbyPOIs, RespondNearbyPOIs, RequestPOIsInfo, RespondPOIsInfo
Permissions: read GIS.
Responsibilities:
Liveness:
TRAVELGUIDE = RegisterDF. ([FindRoutes] || [ProximitySearch] || [CreateMap] || [GetPOIInfo]) $^{\omega}$
FINDROUTES = RequestRoutes. InvokeGetRouteGISFunction. RespondRoutes
PROXIMITYSEARCH = RequestNearbyPOIs. InvokeGetNearbyPOIsGISFunction. RespondNearbyPOIs
CREATEMAP = RequestMap. InvokeGetMapGISFunction. RespondMap
GETPOISINFO = RequestPOIsInfo. InvokeGetPOIsInfoGISFunction. RespondPOIsInfo
Safety: A successful connection with the GIS is established.

Figure 17. The Gaia roles model.

RequestMap		
PersonalAssistant	TravelGuide	
Ask for a map		The Map request includes the coordinates defining a rectangle along with the desired displayed POIs.

Figure 18. The Gaia interactions model

Furthermore, during the analysis phase, the possible interactions with a role's external environment are identified and documented in the environmental model. There, the possible actions that the role can perform to the environment along with the perceptions that it can receive are identified. It is a computational representation of the environment in which the MAS will be situated.

Finally, the rules that the organization should respect and enforce in its global behavior are defined. These rules express constraints on the execution activities of roles and protocols and are of primary importance in promoting efficiency in design and in identifying how the developing MAS can support openness and self-interested behavior.

In a next phase, namely the architectural design phase, the roles and interactions models are refined and finalized by the definition of the system's organizational structure in terms of its topology and control regime. This activity involves considering the organizational efficiency, the real-world organization in which the MAS is situated, and the need to enforce the organizational rules.

Lastly, the Gaia detailed design phase, maps roles into agent types and specifies the right number of agent instances for each type. Thus, an agent type is an aggregation of one or more agent roles. A sample Gaia Agent model is shown in Figure 19, where the agent types "EventsHandler" and "PersonalAssistant" are defined; each integrating the like-named role and the "SocialType" role. However, Gaia does not show how this integration is done in the implementation level.

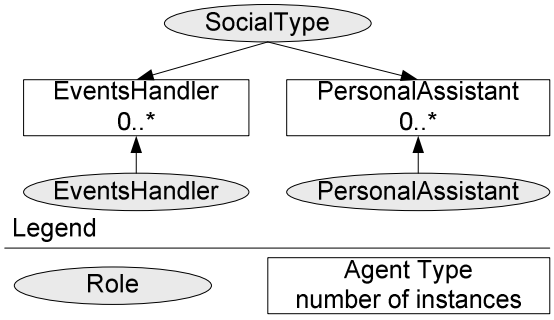


Figure 19. The Gaia Agent model

Moreover, during this phase, the services model, the services that a role fulfils in one or several agents, is described. A service can be viewed as a function of the agent and can be derived from the list of protocols, activities, responsibilities and the liveness properties of a role.

The FIPA Methodology Technical Committee (Garro et al, 2004) defined the process of analyzing and designing a MAS using Gaia by employing the Software Process Engineering Metamodel (SPEM), a standard developed by the Object Management Group (2002).

Gaia, however, has specific limitations related to its use as a complete software development methodology. It does not commit to specific techniques for modeling, nor does it provide guidelines for code generation. The "services model" of Gaia does not apply to modern agents who provide services through agent interaction protocols. Furthermore, the protocol model of Gaia does not provide the semantics to define complex protocols and the Gaia2JADE process additions remedied this situation only for simple protocols. Moreover, Gaia does not explicitly deal with the requirements analysis phase; however, in Zambonelli et al. (2003) the authors propose that it could be integrated with goal-oriented approaches.

The Gaia2JADE process

The Gaia2JADE process (Moraitis and Spanoudakis, 2006), which was developed as a preliminary result of this thesis, is concerned with the way to implement a multi-agent system with the JADE framework (Bellifemine et al., 2001) using the Gaia methodology for analysis and design purposes. It is not presented here in detail as ASEME incorporates all its advantages. A preliminary version of the Gaia2JADE process was presented by Moraitis et al. (2003a). This process is particularly dedicated to the conversion of Gaia models to JADE code. It is described using the Software Process Engineering Metamodel (SPEM) and extends the one proposed by FIPA for describing the Gaia modeling process (Garro et al, 2004). Thus, it proposes to potential MAS developers a process that covers the full software development lifecycle. The Gaia2JADE process has been used for implementing real world multi-agent systems conceived for providing e-services to mobile users (Moraitis et al., 2003b; Moraitis et al., 2005).

This process used the Gaia models and provided a roadmap for transforming Gaia liveness formulas to Finite State Machine diagrams and then provided some code generation for JADE implementation. It also proposed some changes to Gaia such as the incorporation of a functionality table, where the activities were refined to algorithms, and a way to describe simple protocols. For example, in Figure 20, the RequestMap interaction is connected to a RespondMap interaction showing that it must follow the first in order to define the CreateMap protocol.

However, the aim of the authors was not to promote the use of Gaia methodology against other existing methodologies, but to show how one who decided for his own reasons, to use Gaia for the analysis and design phases, can use JADE for the implementation phase. This extension allowed for easily conceiving and implementing relatively simple agents. Finally, its models cannot be used for

simulation-optimization. The reader is directed to Moraitis and Spanoudakis (2006) for the detailed Gaia2JADE process presentation.

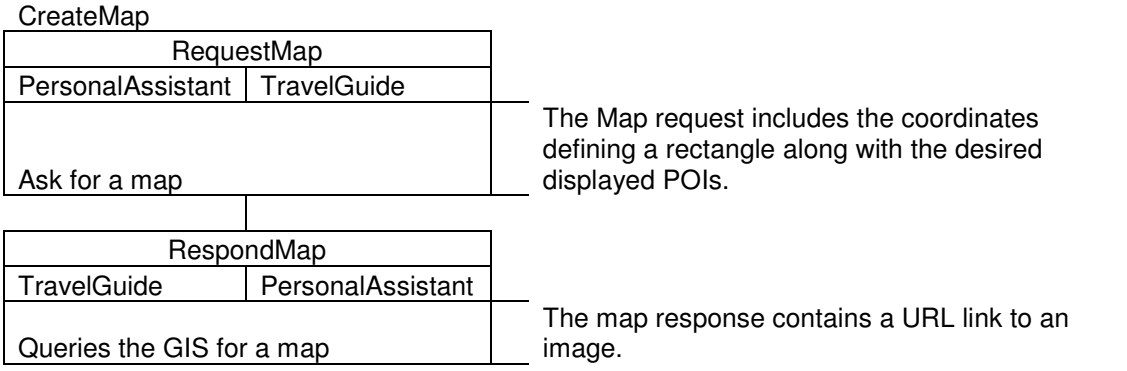


Figure 20. A Gaia extended interactions model

2.2.3 Agent UML

Agent UML (AUML) started as a way to represent agent interactions by extending UML (Odell et al., 2000). It evolved to a complete method for building agent systems (Odell et al., 2001) and, later, it became compatible with UML 2.0 (Bauer and Odell, 2005). This is why AUML is presented in this section even though several researchers did not consider it as a methodology but more like an infrastructure or tool (see Bergenti et al., 2004).

AUML's main contribution is the protocol model that allows to design inter-agent protocols and which was adopted by FIPA. FIPA proposed several extensions to UML 1.x version (i.e. roles, decision points, concurrency, modularity and multi-casting), some of which were implemented in the later 2.0 version (loops, alternatives, parallelism). In Figure 21, a sample AUML protocol model is presented for modeling the contract net protocol (Smith and Davis, 1981) in the UML 1.x with the extensions proposed by (Odell et al., 2001) and in UML 2.0.

Figure 21 shows the semantics for modeling a decision point in the sequence resulting to one or more alternative possibilities. For example, in the contract net protocol (CNP) an initiator sends a call for proposals (*cfp*) message to all participants. Each participant can respond either with a *refuse* or with a *propose* message. The receipt of each of these messages by the initiator initiates a different activation box. Activation boxes are the opaque white rectangles drawn on top of the lifelines that represent each role and they represent that processes are being performed by that role in response to the received message. For example, the lifeline of the initiator role in Figure 21(a) has six activation boxes, all but the first initialized by a received message. However, using this notation can make a protocol definition very complex

especially in the case that multiple rounds of proposals take place or in the case that many different roles are involved.

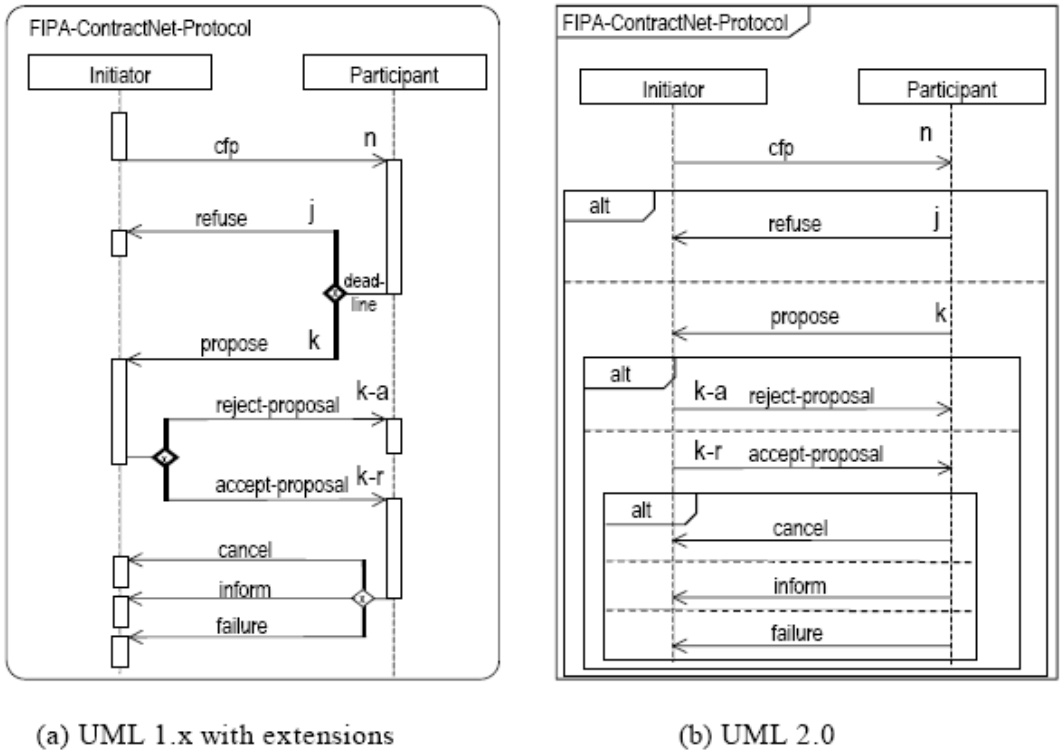


Figure 21. UML 1.x agent extensions and UML 2.0 Sequence Diagrams in AUML (Bauer and Odell, 2005)

According to AUML, modeling MAS can be a top-down decomposition process starting from the roles and protocols. Thus, in Figure 22, the reader can see how an activation box in a protocol model can be further elaborated using other AUML protocol models or standard UML diagrams such as activity diagrams. However, AUML does not describe how these models can relate to each other or to implementation. Neither does it describe how to integrate different roles in a single agent.

AUML allows the actors in the UML use case diagram to be included in the system box representing agents. Moreover, they modify the association type between actors and use cases to represent the number of messages exchanged and their direction (from the sender towards the receiver). The authors demonstrate this AUML use case diagram in Figure 23 showing the use cases between an *Order Handler* and a *Customer*.

AUML has been proposed as a language for modeling multi-agent systems. However, it does not come along with a methodology or a complete process for software development. Many methodologies, i.e. Tropos, Mas-CommonKADS, PASSI, ADELFE and MESSAGE (Henderson-Sellers and Giorgini, 2005), use some of its models, mainly

the agent interaction protocol (AIP) model. The latter has been defined as an extension to the UML sequence diagram.

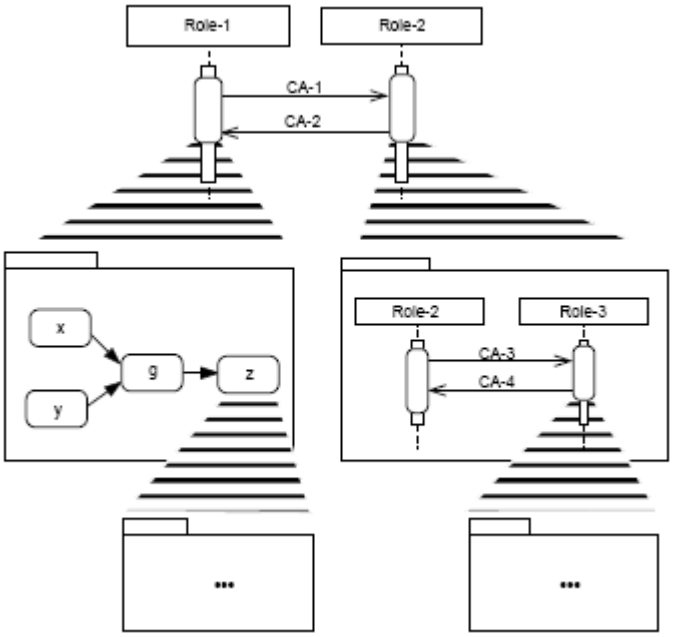


Figure 22. AUML Interaction protocols can be specified in more detail (i.e., leveled) using a combination of diagrams (Odell et al., 2001).

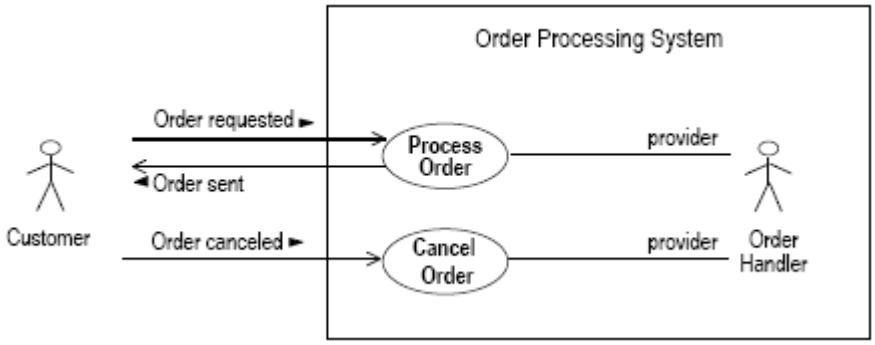


Figure 23. An AUML Use Case Diagram for an Order Processing application (Bauer and Odell, 2005).

However, AIP has specific shortcomings when it comes to defining complex protocols (also see Paurobally et al., 2004). The most important ones are the following:

- the decision points of the participants are not obvious. Only message exchanging is modeled
- there are no semantics for expressing time-dependent concepts like timeouts

- it does not allow the designer to easily model a group that participates in a protocol, but whose members can choose individual actions. In the latter case the designer must include all possible group members in the diagram

The AUML layered approach to protocols provides a mechanism for specifying the program that implements a protocol but does not specify how it is integrated with other such programs (other protocols), or how to integrate it with the other agent capabilities.

2.2.4 Vowels

The Vowels methodology and the Volcano respective multi-agent platform (Ricordel and Demazeau, 2002) is one of the first approaches to engineering multi-agent systems. The main idea of the vowels methodology is that a MAS is consisted of four major component types (each corresponding to a Latin vowel), a) the Agent, b) the Environment, c) Interactions, and d) Organization. It is a methodology that introduced these four different aspects in MAS development for the first time in a modular architecture.

Different design techniques can used to analyze and design each component type. Agents can range from simple automata to complex knowledge-based systems. The environment is usually a model of the real world on which physical agents act (e.g. robots). Interactions can be either message-based or blackboard-based or even based to effects on the environment (an aspect not really addressed even by later methodologies). Organizations can be static or dynamic ones following hierarchical or market-like structures.

The component types are also called *bricks* and are interconnected through another kind of brick, the *wrapper*. The wrappers are used in order to resolve incompatibilities between models. They add flexibility to the MAS model, however they impose a constraint to the developer to define a wrapper for each brick to which he wants to connect an existing one. The methodology aims to the creation of a large number of bricks and wrappers thus facilitating the development of future MAS. That is why the methodology urges developers to define their new bricks to be as generic as possible. Moreover, this approach also creates a big overhead for the engineer that wants to replace an existing brick with a new one having to implement new wrappers for all the bricks connected to it (see Briot et al., 2006).

The different phases of the vowels methodology are presented in Figure 24. The analysis phase consists of two steps. During the first step, a domain ontology is created for describing the information that will be used for defining the problem. The second step is about giving a precise solution to the problem in an implementation independent manner. In the design phase, the engineer chooses the possible orientation of the application towards a specific vowel (brick type), then chooses the model of each brick and the needed wrapper bricks. Then, in the development phase, the bricks are created (programmed or chosen among existing

ones). Finally, during the deployment phase the MAS is deployed using a specific language that describes what building blocks will be deployed.

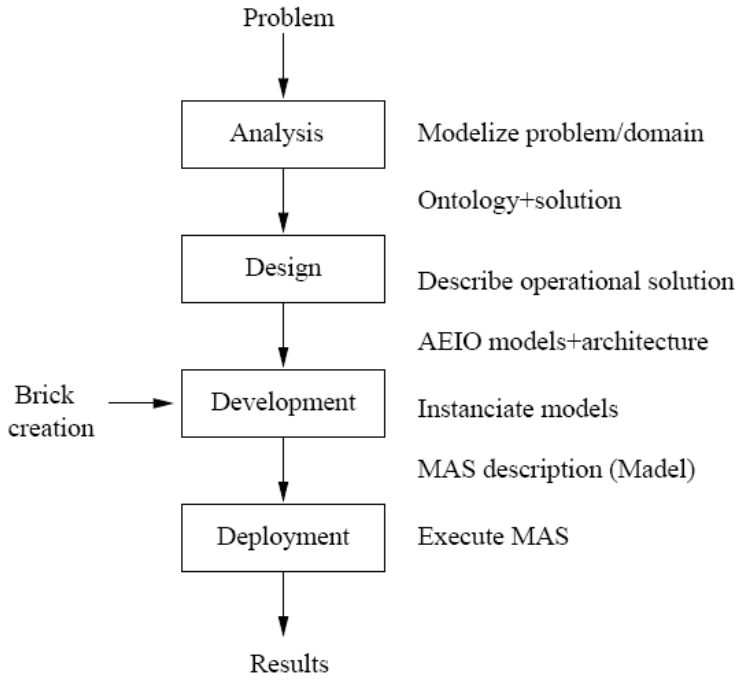


Figure 24. The vowels development phases (Ricordel and Demazeau, 2002).

2.2.5 PASSI

PASSI (Burrafato and Cossentino, 2002; Cossentino, 2005) is an AOSE methodology that aims to allow engineers experienced in UML to model and implement agent-based systems. Thus, all the models that they define are derived from UML models. The PASSI methodology is summarized in Figure 25 where the five phases of the methodology along with the models related to each one of them are depicted.

In the *Domain Requirements Description* model the modeler identifies the system use cases (see Figure 26). In the agent identification phase PASSI splits the traditional UML system box (the one that includes all system use cases) to different boxes grouping the different agents' use cases (see the *Agent Identification Diagram* in Figure 27). The six different boxes represent six different agent types and the use case dependencies between them are labeled as «communicate».

The roles identification phase is about creating extended UML sequence diagrams. PASSI defines that each object in the sequence diagram represents an agent's role with the convention that the objects are named as *<role_name>:<agent_name>* (see a sample roles identification diagram in Figure 28). However, this convention does not allow the participation of more than one instances of a role of a specific agent

type in a scenario (e.g. for defining a scenario where a manager agent broadcasts a request for proposals to many, e.g. task agents).

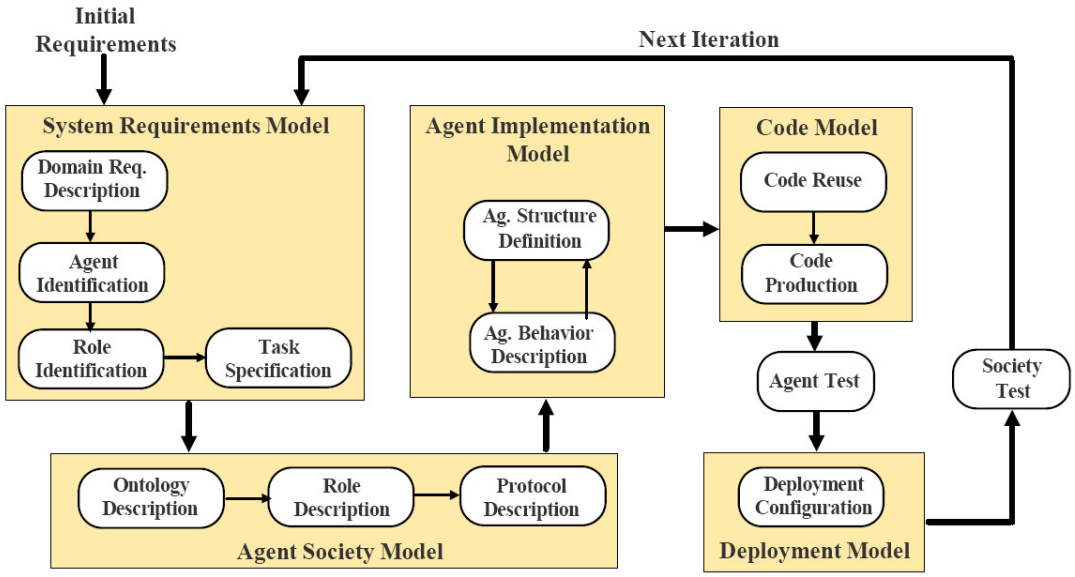


Figure 25. The models and phases of the PASSI methodology (Cossentino, 2005).

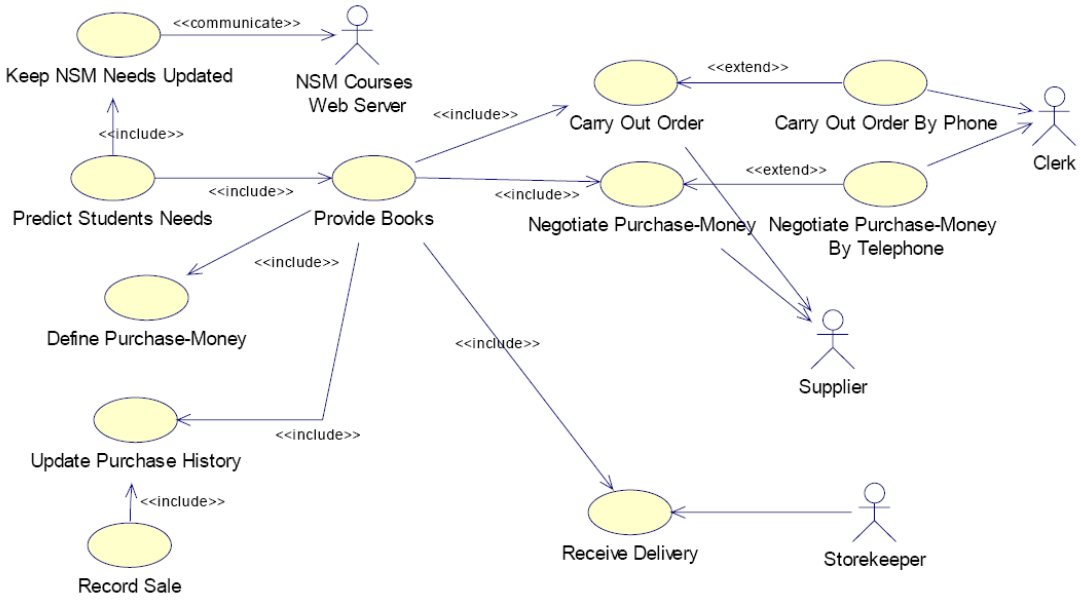


Figure 26. The domain requirements description diagram of PASSI (Cossentino, 2005).

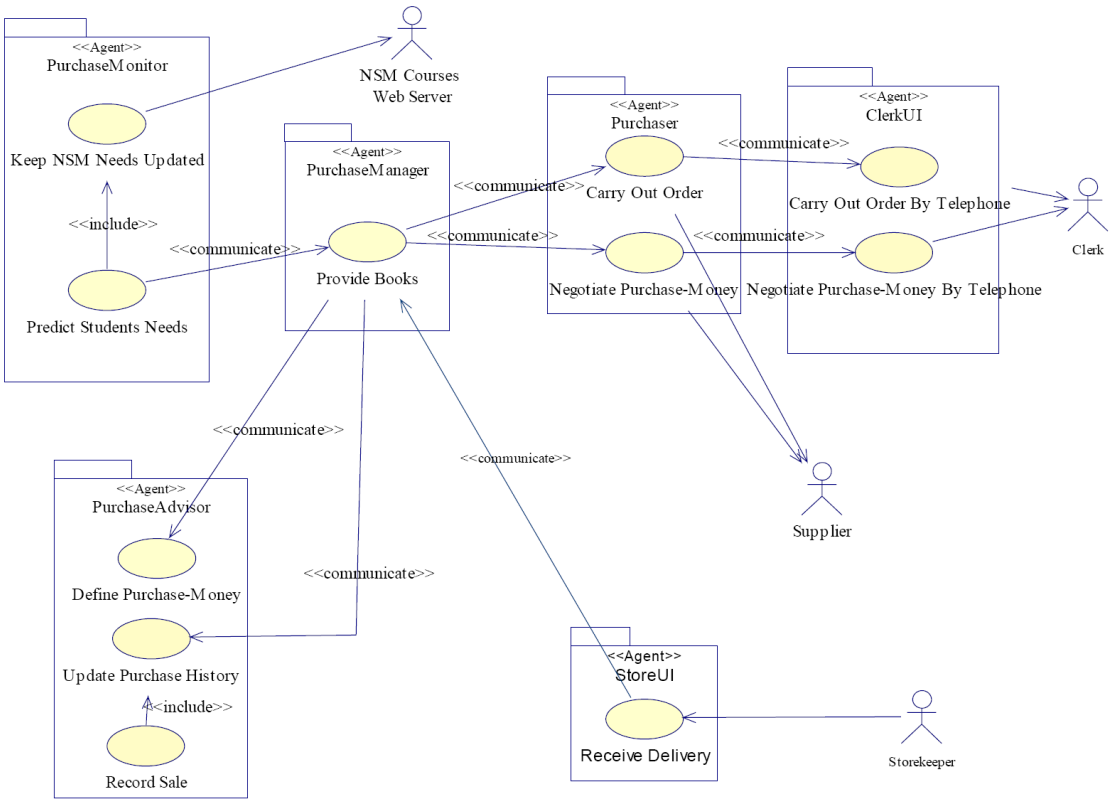


Figure 27. The agents identification diagram of PASSI (Cossentino, 2005).

Then, in the task specification phase a UML activity diagram is created for each agent showing two swimlanes, the first (the one on the left side in Figure 29) containing the tasks of other agents that send or receive messages to or from the tasks of the agent in question (e.g. the Purchase Manager agent on the right side in Figure 29).

The next three models in the Agent society model extend the UML class diagram to define an ontology (according to FIPA standards) the roles of the agents (as classes associated with the realized protocols with arrows from the initiator to the responder) and the protocols descriptions (usually through AUML AIP diagrams). The FIPA defined protocols are built in allowing the developer that is satisfied by one of them to select it.

The Agent Implementation model phase iterates between the *Agent Structure Definition* and *Agent Behaviour Definition* models in two levels, the multi-agent and the single agent one. They are static views (extended UML class diagrams), the Agent Structure Definition depicting the agents with the possible association paths (in the multi-agent structure definition) and with the tasks of an agent (in the single agent structure definition). The same holds for the Agent Behaviour Definition, in the multi-agent level showing the tasks of all agents in a UML activity diagram and in the agent level only one agent's tasks.

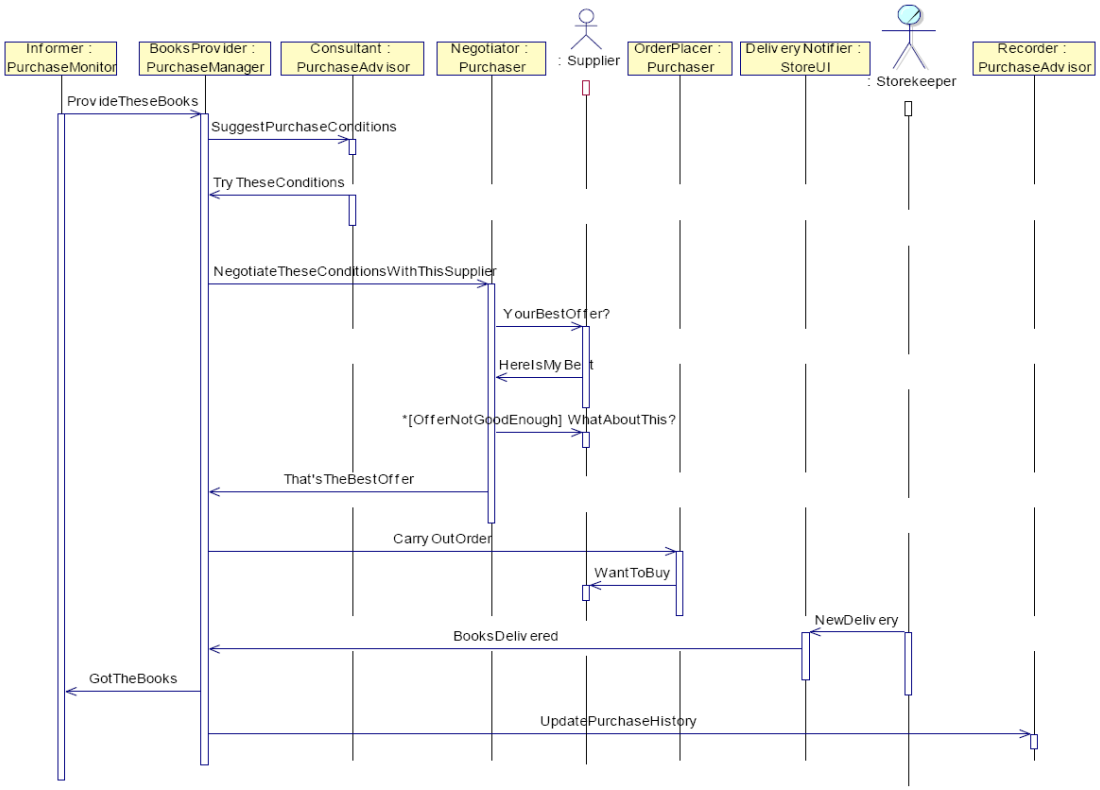


Figure 28. A PASSI Roles Identification Diagram (Cossentino, 2005)

In the code model the developer can choose among ready implementations of FIPA protocols and previously developed code to associate with agent's tasks, helped by the PASSI PTK tool and a specific AgentFactory application that reads the class diagrams of the previous level (Chella et al., 2004).

Chella et al. (2006) proposed an agile version of the PASSI methodology in which they use tools for allowing patterns reuse and automatic production of parts of the design documentation. In their work they allow for agile development using only half the artifacts of the PASSI methodology.

All in all, PASSI starts immediately in use case description omitting the stakeholders and goals identification phase. PASSI extends the UML use case diagram notation and semantics in a way not easily apparent to a modeler that is familiar with it. Then again, the scenarios (or AUML AIP models) are used by the engineer in order to produce the task specification diagram without a clear transformation technique.

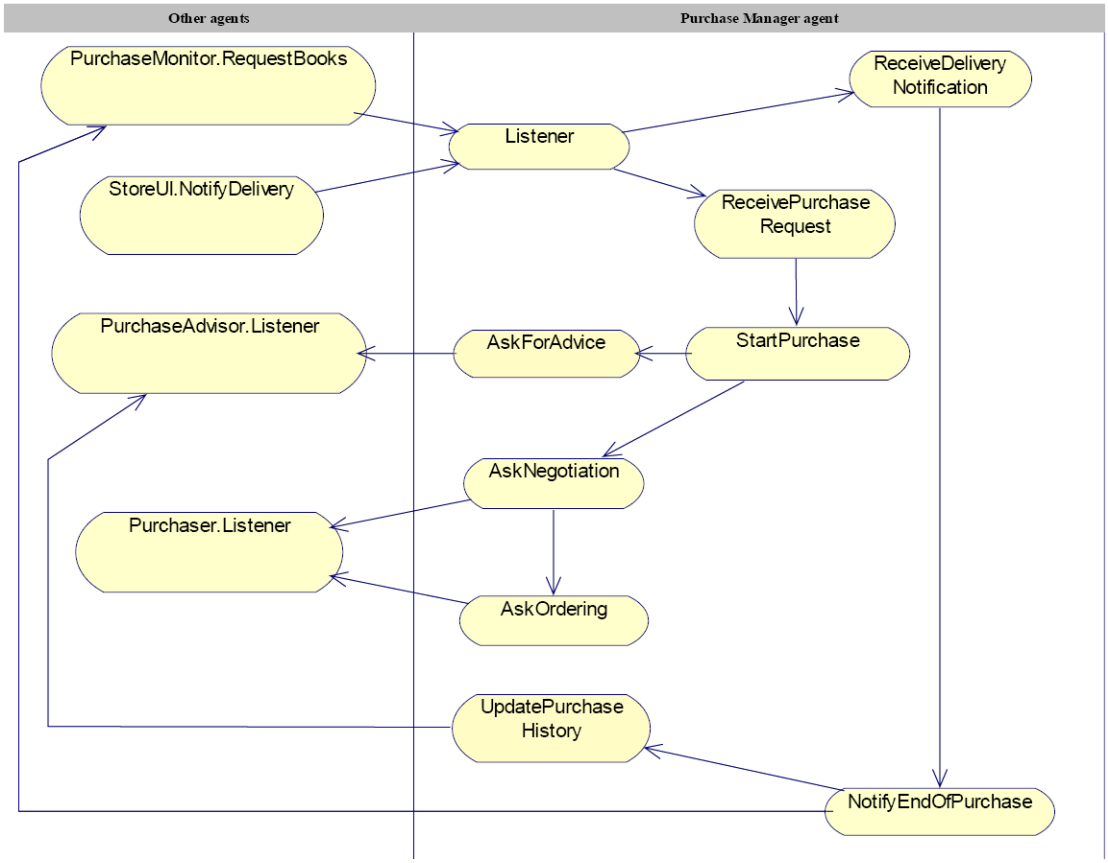


Figure 29. A PASSI Activity Diagram (Cossentino, 2005).

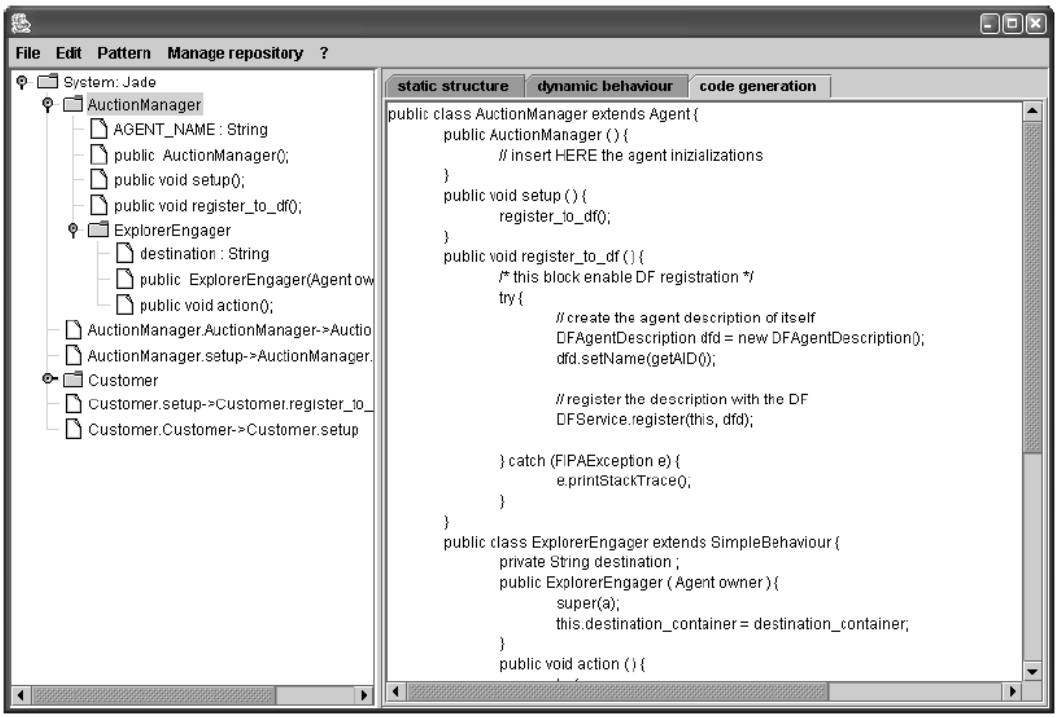


Figure 30. A screenshot from the AgentFactory tool (Chella et al., 2004)

2.2.6 Prometheus

The Prometheus methodology has been proposed by Padgham and Winikoff (2003 and 2004). It provides a method and a process for developing multi-agent systems. Prometheus supports the development of *intelligent* agents linking the word intelligence with the analysis and design of an agent as an entity with goals, beliefs, plans and events. It uses the JACK Intelligent Agents Platform (Winikoff, 2005) for system implementation that is also centered on the definition of these terms. It has been conceived as a methodology that will be used by non-experts, including undergraduate students.

Prometheus defines three phases; a) system specification, b) architectural design and c) detailed design (see an overview of the methodology phases and work products in Figure 31). During the first phase the environment to which the system under development will be situated is defined along with the goals and functionality of the overall system. The environment is defined as a series of events that can be perceived by the system (percepts) and a series of actions that the system will be able to execute. Then the modeler defines the system goals, the functionality needed to achieve these goals and use case scenarios that show sequences of interleaved actions, percepts, and exchanged messages.

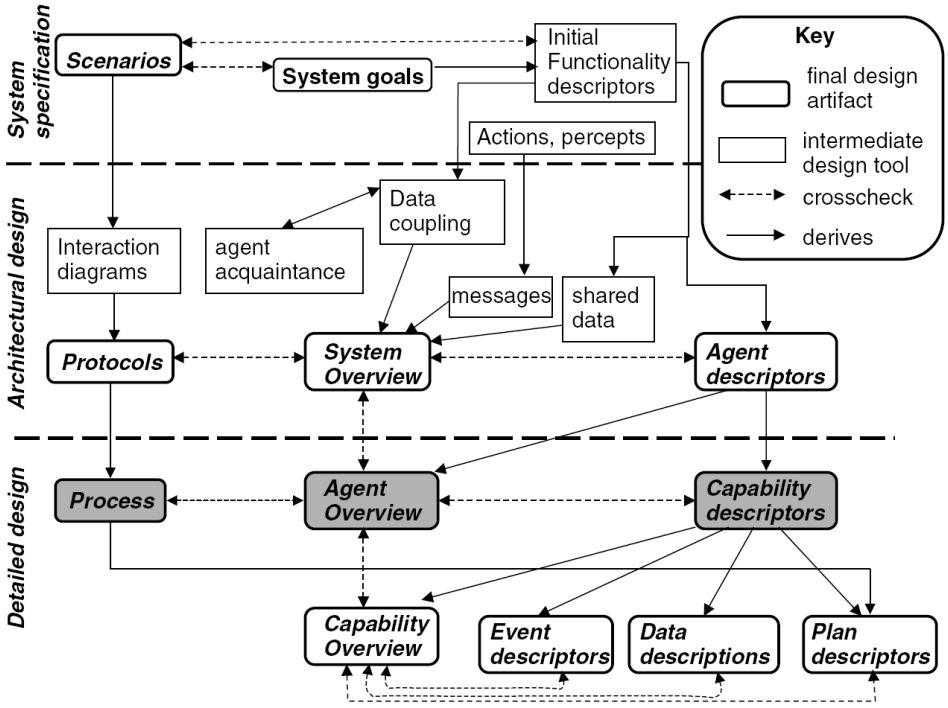


Figure 31. The Prometheus phases and work products (Padgham and Winikoff, 2004).

The second phase (architectural design) defines three activities; firstly, the agent types are determined by grouping functionalities. Each agent type is assigned an agent descriptor that includes these functionalities, information about when and how the agent is instantiated and destroyed, the data, percepts and actions related to it and, finally, the agents that it interacts with. Then during the second activity, the system overview diagram is created. It shows the agents types, the possible interactions, the data handled by each agent type, the possible messages that an agent can send, the actions and percepts of the whole system and the agents related to each of them. The system overview diagram can be seen as a static view of the system. In Figure 32 an example of a system overview diagram is presented along with an explanation of the different icons used for drawing it. The agents are connected with the different message types that they exchange. They are also associated with percepts, data and actions.

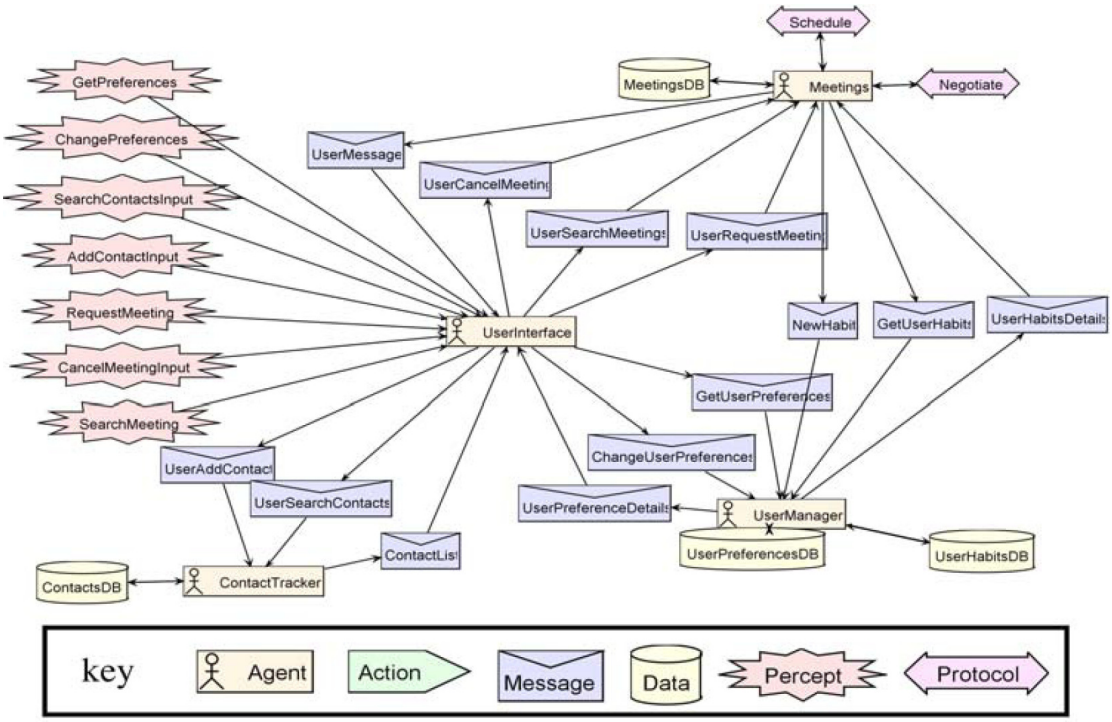


Figure 32. Prometheus: Example of a system overview diagram (Padgham and Winikoff, 2005)

The third activity of the architectural design phase defines the dynamic view of the system as valid sequences of messages exchange between the different agent types. Towards this end, the AUML agent interaction diagrams are employed. In Figure 33 the protocol descriptor template is presented. Each protocol has a name, a description, one or more messages involved, the scenarios of the previous phase to which it corresponds, the names of the involved agents and a notes field, where the AIP diagrams are placed.

Protocol Descriptor
Name: Description: Included Messages: For each indicate the source and destination, e.g. request (AnAgent→ AnotherAgent) Scenarios: Agents: Notes:

Figure 33. The Prometheus protocol descriptor template

The detailed design phase focuses in the agent level. Thus, the *agent capabilities* are defined as the events that can be generated and received by the agent. Moreover, other elements such as *internal events*, *plans*, and *detailed data structures* are defined for each agent type and depicted in the agent overview diagram. These elements correspond to JACK agent code. In Figure 34, the reader can see a sample agent overview diagram for the *Meeting agent*. It has some similarities with the system overview diagram, however here the agent capabilities replace the agents. One issue that will surely draw the reader's attention is the association between capabilities with messages (internal messages).

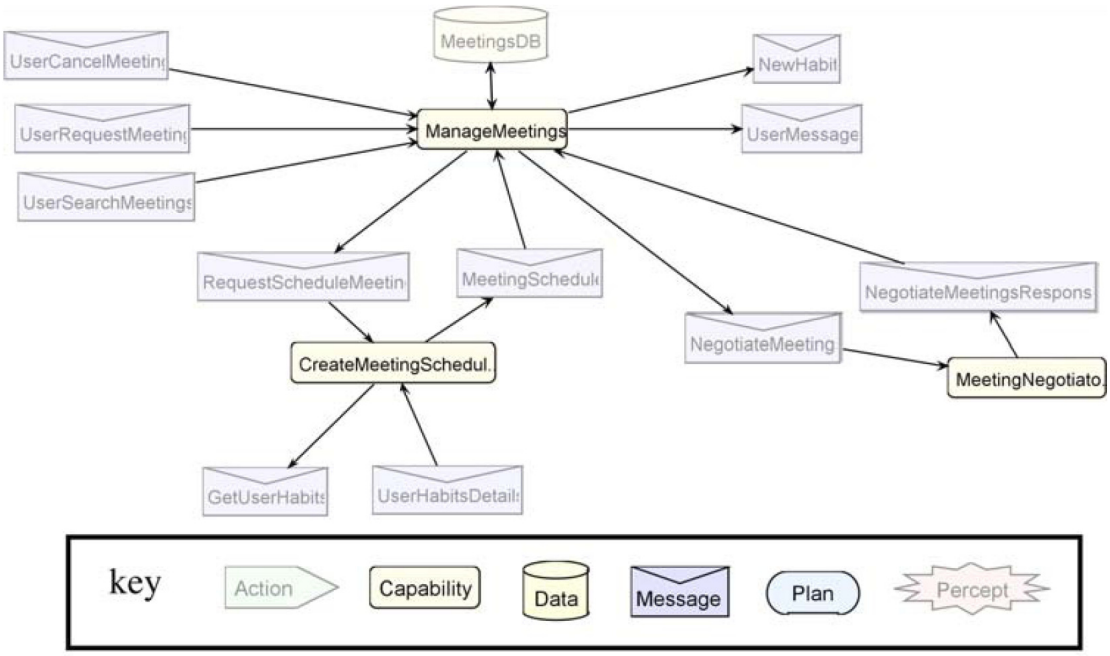


Figure 34. Prometheus: Example of an agent overview diagram: Meeting agent (Padgham and Winikoff, 2005)

In Prometheus the authors use the terms of functionality and capability. However, they are not used as independent terms. In fact, functionalities and capabilities refer

to the same concept as it evolves through the development phases (i.e. the abilities that the system needs to have in order to meet its design objectives). The support for implementation, testing and debugging of Prometheus models is limited and it has less focus on early requirements and analysis of business processes (Henderson-Sellers and Giorgini, 2005).

Another limiting issue of the methodology is the fact that the protocols definition using AIP diagrams is not used later somehow formally at the agent level. This means that the developer has to undertake the mental task of transforming the AIP diagrams to processes. In their book, Padgham and Winikoff (2004) propose that process diagrams are to be developed by looking at the protocols involving the agent in question, as well as the scenarios developed and the goals of the agent. This contradicts the overview diagram shown in Figure 31 and is an issue that almost all the AOSE methodologies suffer from, the lack of a systematic way to integrate interaction protocol specifications to the agent capabilities.

2.2.7 Ingenias

Ingenias (Pavón and Gómez-Sanz, 2003) is a methodology that emerged with a development environment allowing for agent development using the Ingenias metamodel. Its metamodel is the richest one among AOSE methodologies containing more than 300 concepts (the ecore[1] metamodel of Ingenias can be downloaded from http://ingenias.sourceforge.net). This feature can also be considered as exceptionally restrictive to developers that want to use their own agent architectures but also needing more learning time than all other methodologies in order to begin working with it. Its process is also hard to learn and use (especially in iterative development) as it consists of about 100 activities (Pavón et al., 2005).

It defines a whole new set of models and associates them with UML models aiming to define the concepts relevant to agent development and ground them to UML for helping the development phase with an object oriented language. The reader can only take a taste of the INGENIAS diagrams (this thesis cannot go in detail to this methodology as it would be very lengthy) in Figure 35. The agent viewpoint describes the functionality of an agent in terms of goals, tasks and capabilities (or roles it plays). These are captured by the following concepts:

- The *Mental State* includes all the information needed for the decision making processes of an agent. This information is the agent's goals, beliefs and facts.
- The *Mental state manager* (M) provides operations for creating, deleting and modifying Mental State entities.
- The *Mental state processor* is responsible for deciding which task to execute among the agent's tasks.

[1] See Chapter 5 for the definition of the ecore metametamodel.

INGENIAS clearly distinguishes between an agent and an application showing that agent technology isn't about substituting existing frameworks, for example for building user interfaces but for adding new characteristics to computer systems. Agents access applications through a kind of Application Programming Interface (API) that they offer. An issue that INGENIAS leaves to the developer is whether he will define first the tasks or goals of an agent. Maybe this is the result of the lack of a requirements analysis phase. Moreover, Ingenias does not offer the convenience of gradually modeling a multi-agent system by considering it at different levels of abstraction.

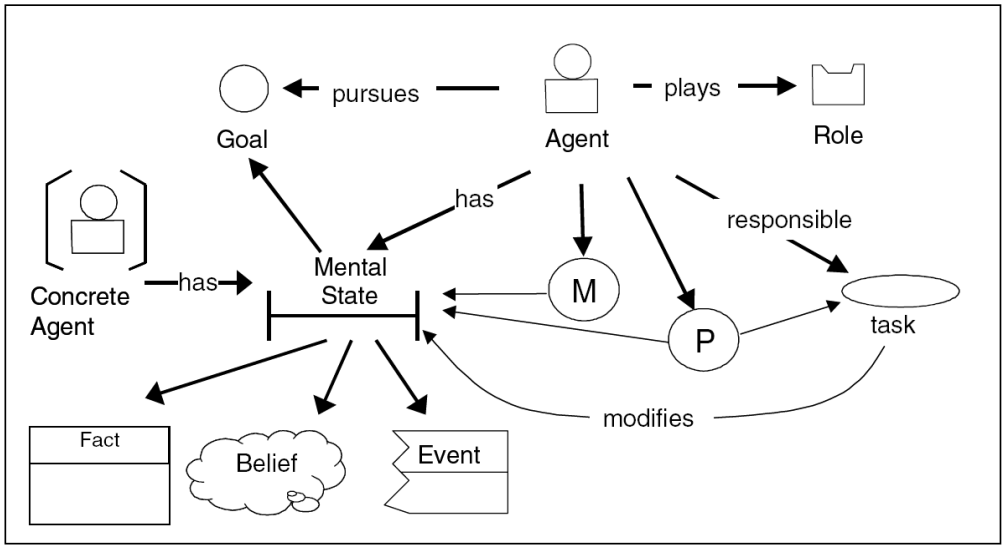


Figure 35. Elements of the agent viewpoint in INGENIAS (Pavón et al., 2005).

García-Magariño et al. (2009) in their original work present an algorithm to generate model transformations by-example. This algorithm facilitates the generation of many-to-many transformations between arbitrary graphs of elements; dealing with transformation languages that do not directly support graphs of elements in their source or target models. They developed the MTGenerator tool for the application of the algorithm to support the agent-oriented software processes of the INGENIAS methodology, which implements the algorithm for the ATLAS transformation language.

Their approach allows the engineer to define himself the transformations that he wants to apply to models complying with the INGENIAS metamodel. Taking into account the huge Ingenias metamodel and the many possible paths that the engineer can follow, this solution on one hand gives a freedom to the engineer but burdens him with the additional work to define the transformations himself.

2.2.8 Tropos

TROPOS (Bresciani et al., 2004) is a methodology whose main difference with other methodologies is its focus in the early requirements analysis phase where the actors and their intentions are identified in the form of goals. The latter are divided in two categories, hard goals (related to functional properties of the actors) and soft goals (related to non functional properties of the actors). Actor diagrams depict the actors, their goals and dependencies on other actors for realizing a goal. Then, goal diagrams analyze the goals of a specific actor to subgoals and plans for achieving the goal. In the late requirements phase the models are extended adding possible interactions between goals (helpful or conflicting goals).

Thus, a sample actor diagram is presented in Figure 36 for a media shop. The main actors are Customer, Media Shop, Media Supplier, and Media Producer. The Customer actor depends on the Media Shop actor to fulfill the goal “Buy Media Items”. The Media Shop actor depends on the Customer actor for its softgoals “Increase Market Share” and ”Happy Customers”. The Customer also depends on Media Shop to fulfill the task “Consult Catalogue”. Likewise, there are dependencies between the Media Shop and Media Supplier actors and between the Media Supplier and Media Producer actors to complete the value chain.

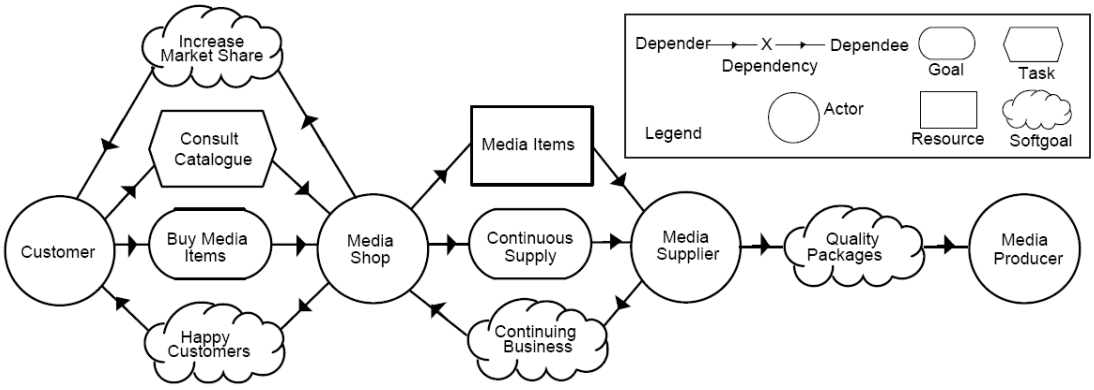


Figure 36. Actor diagram for a Media Shop (Giorgini et al., 2005)

In the late requirements analysis the selected for implementation actor(s) plans and goals are refined. The relevant model for the media shop and specifically its Medi@ actor is presented in Figure 37. In this model the analyst can find tasks decomposition like in the case of “Shopping Cart” that is achieved by the subtasks “Select Item”, “Add Item”, “Check Out” and “Get Identification Detail”. The definition of a task that contributes to a softgoal is denoted by a an association towards the softgoal with plus or minus signs indicating a positive or negative influence.

The architectural design phase is a three step process that starts by including new actors in an extended actor diagram. In the next step the capabilities of each actor

are identified and finally they are grouped to agent types. A suggested approach to defining actors is the *Structure-in-5,* which specifies that an organization is an aggregate of five sub-structures:

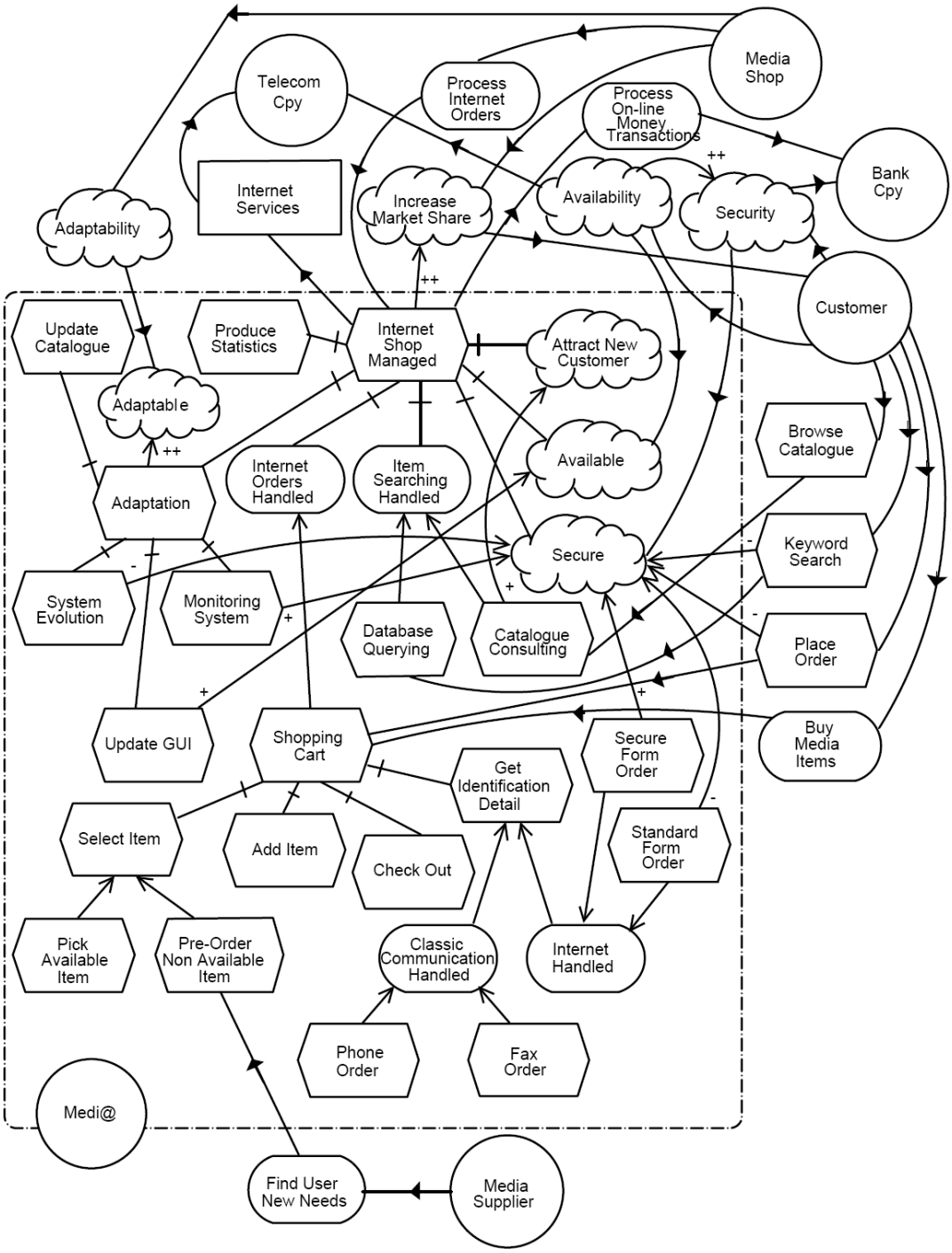


Figure 37. The late requirements analysis model for the electronic media shop Medi@ (Giorgini et al., 2005).

a) the *Operational Core* at the bottom, which carries out the basic tasks and procedures directly linked to the production of products and services,

b) the *Strategic Apex* at the top, which makes executive decisions ensuring that the organization fulfills its mission in an effective way and defines the overall strategy of the organization in its environment,

c) a list of *managers* in the middle responsible for supervising and coordinating the activities of the Operational Core. Such are also the *Technostructure* (for adapting the organization to the operational environment and standardizing procedures) and the *Support* (providing services not in the business core such as a cafeteria) that influence the operating core only indirectly.

A structure-in-5 analysis for the Medi@ is presented in Figure 38. The Decision Maker actor corresponds to the Strategic Apex role, the Store Front to the Operational Core role and the Back Store to the Support role (providing accessory services such as creating a back-up for the database). Finally, the Coordinator and Billing Processor act as managers.

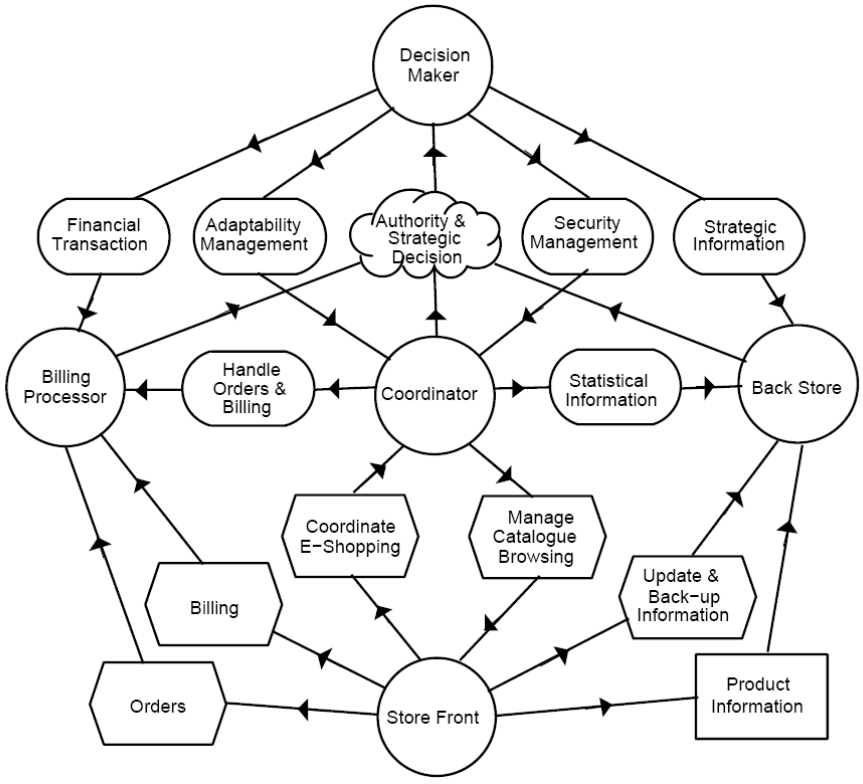


Figure 38. The Medi@ architecture in Structure-in-5 (Giorgini et al., 2005)

The next phase, detailed design, is concerned with modeling the capabilities and plans of the agents using UML activity diagrams and the agents' interactions using AUML interaction diagrams. Finally, in its implementation phase, Tropos provides some heuristics and guidelines for mapping Tropos concepts to BDI concepts, which

can themselves be mapped to JACK constructs for implementation. Figure 39 shows the suggested decomposition of the Store Front actor based on several existing patterns such as the booking pattern (between the Shopping Cart and the information broker), or the matchmaker pattern (the “Source Matchm.” locates the appropriate source for the Info Broker).

TROPOS provides a formal language and semantics that greatly aid the requirements analysis phase. It is a process centric design approach and the detailed design phase of TROPOS proposes the use of AUML. Finally, Tropos has been applied for modeling relatively simple agents, not complex ones (Henderson-Sellers and Giorgini, 2005).

An MDA-compliant work based on Tropos has been presented by Perini and Susi (2006), where the authors define rules for transforming a Tropos plan decomposition diagram to a UML activity diagram. They present their rules formally and show how they can build a tool for applying these rules automatically. However, they do not tackle the issue of transforming an AIP diagram to a plan.

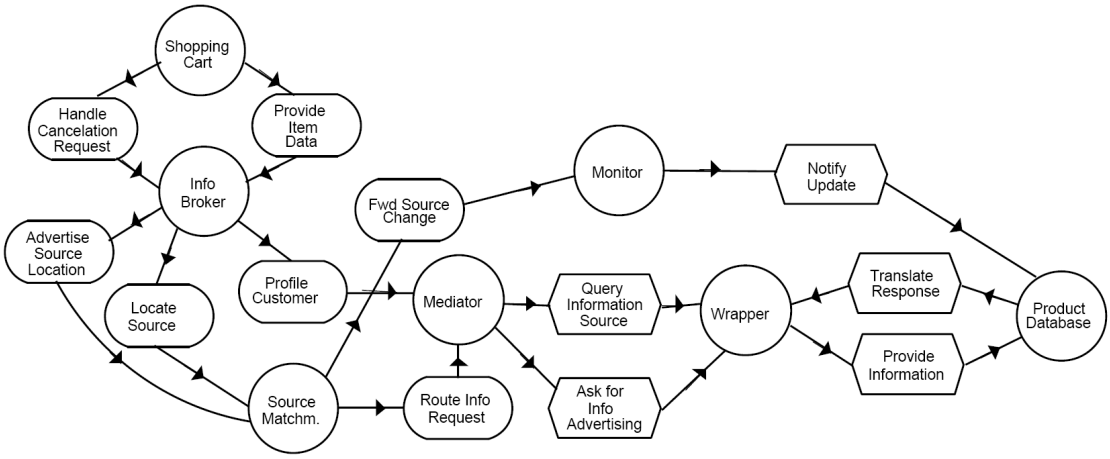


Figure 39. Store Front actor decomposition with social patterns (Giorgini et al., 2005)

Finally, even as Tropos starts by identifying stakeholders and their goals in the requirements analysis phase, it ends up by proposing the development of a system composed by a large number of agents not representing the original actors, but more actors that appear during the tasks decomposition. This is better shown in the case of the example of MEdi@ where the shopping cart (usually a data structure for storing items selected by the user while exploring an electronic store’s web site) is identified as an actor (to be developed as an agent). A classical software engineering architecture would define the shopping cart as a stateful object that is instantiated for a user’s session (see Jacyntho et al., 2002).

2.2.9 Modeling inter-agent protocols

This paragraph will first define what an agent communication language is and then it will focus on the proposal of Moore on conversation policies (as it is an important background for this work). It also discusses other approaches trying to encompass the most popular directions and methods for modeling inter-agent protocols.

2.2.9.1 Agent Communication Language

The term *Agent Communication Language (ACL)* is used for describing any agent communication language. Languages for communicative agents are intended to play the role that natural languages play for their human counterparts (Labrou et al., 1999). Usually, the message types of ACLs (or *performatives*) are understood as *speech acts*. The latter are defined by the Speech Act Theory (SAT).

One of the works that firstly proposed SAT is that of Austin's (1975). A speech act is an act that a speaker performs when making an utterance. Performatives express the intent of an agent when it sends a message to another agent. Thus, a message has four parts, a) the sender, b) the receiver, c) the performative and d) the message content (what is said). For example, the performative “inform” may be interpreted as a request that the receiving agent adds the message content to its knowledge-base. SAT is also accepted by FIPA in defining the communicative acts of the FIPA standard Agent Communication Language (ACL, see FIPA TC Communication, 2002b). A message can be defined by the atom:

performative(sender, receiver, content)

2.2.9.2 Conversation Policies And The Need For Exceptions

Moore (2000) proposes an inter-agent protocol formalism based on statecharts and the Formal Language for Business Communication (FLBC) ACL (Moore and Kimbrough, 1995). For his work on conversation policies, Moore makes the assumption that developers that adopt his models can understand a formal specification and implement it in whatever way they see fit. In the FLBC, Moore defines, for example, that the message *request(sender, receiver, action)* expresses that:

a) The *receiver* believes that the *sender* wants him to do the *action*

b) The *receiver* believes that that the *sender* wants the *receiver* to want to do the *action* (Moore, 1999)

However, some agents might not have the ability (or the need) to model their (or other agents) beliefs and would respond in directly doing the *action*. According to the work of Moore, the conversation policies are implementation independent. A conversation policy (CP) defines:

a) how one or more conversation partners respond to messages they receive,

b) what messages a partner expects in response to a message it sends, and,

c) the rules for choosing among competing courses of action.

A CP is well-formed if it does not contain contradictory directions for what a partner should do. Moore allows a message to interrupt a current conversation when it is neither an expected, nor the standard reply to the previous message. Moore's conversation policies allow for exceptions when a conversation is interrupted by assuming that an agent has stored all allowed CPs in a kind of repository where he can browse a new policy to handle the exception in the form of a *subdialog* to the original one. When this subdialog terminates the original one can resume.

Moore introduces the idea of modeling the activities of the participants in a conversation as orthogonal components of a statechart. In Figure 40 a conversation between a broker agent (represented by the AND-state "asked if appropriate broker for a product") and a provider agent (represented by the AND-state "advertise with broker"). Note that the transition expressions contain the actions of sending and receiving a message.

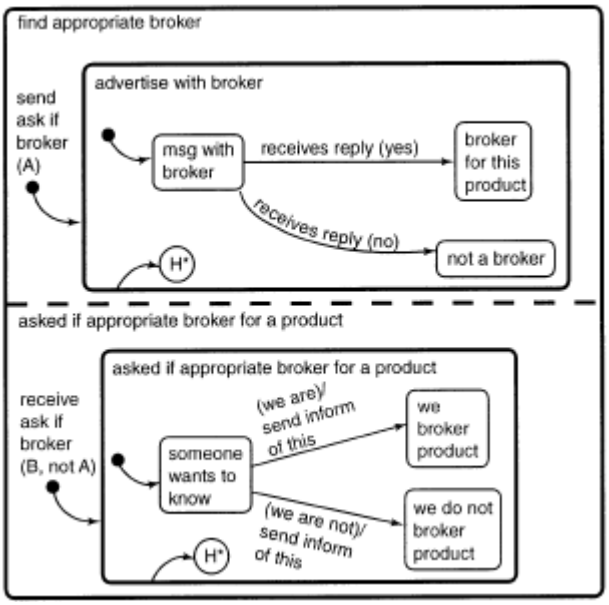


Figure 40. A statechart that describes the activities of both parties in a conversation (Moore, 2000).

In Figure 41, the provider is assumed to be executing the "inform broker about product we sell" conversation when an inform message arrives. This message is not expected, but has the same *conversation id* with the currently executing conversation. Conversations are assumed to have a unique identification string (the conversation id) so that the receiver can identify the relevant conversation to this message (an agent may be concurrently involved to many conversations). In Figure 41 this id is the "conversation823". This inform message is tested against available

CPs. The CP "standard effects for inform" is found in the agent's repository that starts with an inform message. This CP is activated and when it is finished the previous CP resumes.

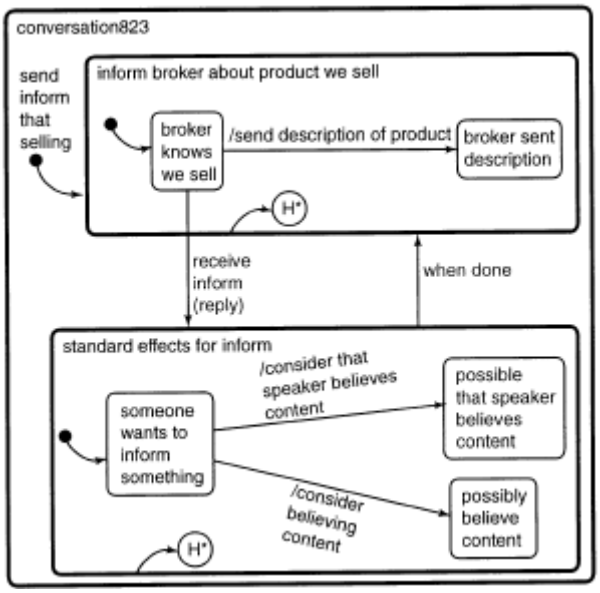


Figure 41. A statechart representation of a conversation policy with an unplanned-for subdialog (Moore, 2000).

2.2.9.3 Other Works

Paurobally et al. (2004) propose that an inter-agent protocol should be:

a) correct (having no contradictory states),

b) unambiguous (defining what each agent should do),

c) complete (defining all possible outcomes) and,

d) verifiable (its properties can be verified).

Recognizing the fact that a protocol should have both a graphical and formal representation they combine the language of statecharts and a language based on Propositional Dynamic Logic (PDL), the Agent Negotiation Meta-Language (ANML). Propositional dynamic logic, or PDL, was derived from dynamic logic in 1977 by Michael Fischer and Richard Ladner. PDL blends the ideas behind propositional logic and dynamic logic by adding actions while omitting data; hence the terms of PDL are actions and propositions.

ANML models agent interaction protocols in the form of multi-modal theories, leading to an abstract theory of an interaction in a group. ANML extends PDL allowing the definition of agent groups, sets of agents, sets of states, ANML formulas

and complex processes. The formulas of ANML model processes and states, for example, the formula *[a]A* means that *A* holds after executing process *a*. Paurobally et al. (2004) examined all the possibilities for graphically modeling an inter-agent protocol and recognized several advantages and disadvantages to each one of them. The most important ones are presented below with the plus sign indicating an advantage and the minus sign indicating a disadvantage:

- AUML (see §2.2.3)
    - \+ The exchange of messages is shown explicitly
    - \+ The process of the interaction over time is explicitly presented through the timelines
    - \- Poses certain difficulties in multi-parties protocols
    - \- There is no way to express time dependent actions such as timeouts
- Petri Nets (see e.g. Mazouzi et al., 2002)
    - \+ Allow concurrency and synchronization
    - \+ They are supported by tools that detect conflicts
    - \- Very hard to read and conceive
    - \- Very difficult to merge, i.e. design the possibility of an agent participating in more than one petri nets
    - \- Poor scalability due to the fact that there is redundancy in repeating the same parts of the protocol for different agent roles
    - \- Limited reusability and abstraction
- Statecharts
    - \+ States and processes can be treated equally allowing an agent to refer and reason about the state of an interaction
    - \+ Statechart notation is more amendable for extension – simple semantics
    - \+ Visual models are easier to conceive and display – engineers familiar with UML can start working with them immediately
    - \- Participating roles are not shown explicitly
    - \- Compound transitions are not shown in detail
    - \- There is a question of completeness

Then, the authors define the templates for transforming the ANML formulas to statecharts, extending the statecharts language in the process. The representation of

all computation is in transitions, while states just describe a situation (where specific conditions hold).

The reader can get a feeling of the modeling of protocols using propositional statecharts (as the authors have named them). The representation can be general (see Figure 42), or specialized for a specific agent participant (see Figure 43). The expressions in the transitions are ANML formulas include actions and conditions.

The proposal of Paurobally et al. (2004) and later by Dunn-Davies et al. (2005) has some issues that are identified as limiting. The first is the representation of all computation in transitions, while states just describe a situation. In another point of view the transitions should respond to events and messages, while states would allow each agent to perform operations dependent on the situation (this way functionality is also identified). Moreover, they do not use the orthogonality feature of the statecharts because they consider that the agents are not subsystems and that in this case they would have to combine parts of interactions between temporally autonomous agents into a pseudo whole. Furthermore, they argue that in a typical interaction protocol the agent states are not independent, as many or all of the agents may be in the same protocol state at any particular time or may be following a similar sub-protocol.

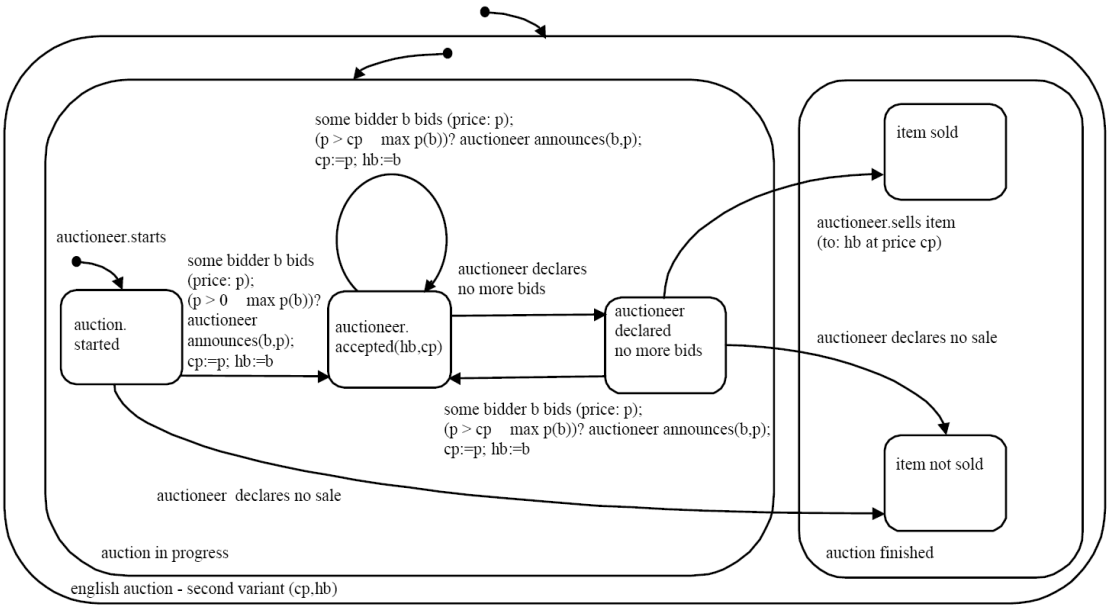


Figure 42. A detailed version of the English Auction protocol with agent/action path event labels (Dunn-Davies et al., 2005)

However, in the view of this thesis, orthogonality is very helpful for providing a complete view of the protocol including all possible actors. Then, when it comes to implementation, each agent type can realize only the orthogonal component that corresponds to his role. Also, using orthogonality, one can develop (and simulate) agents that can concurrently participate into more than one protocols as will be

shown in Chapter 3. Another issue related to their work is the absence of a modeling process for generating the statecharts from specific requirements. Finally, the extended statecharts that they use can be executed only using the Agent Negotiation Meta-Language (ANML). The change in the language of statecharts is so radical that the extended statecharts cannot be used by existing CASE tools and ANML is not used in general by software engineers.

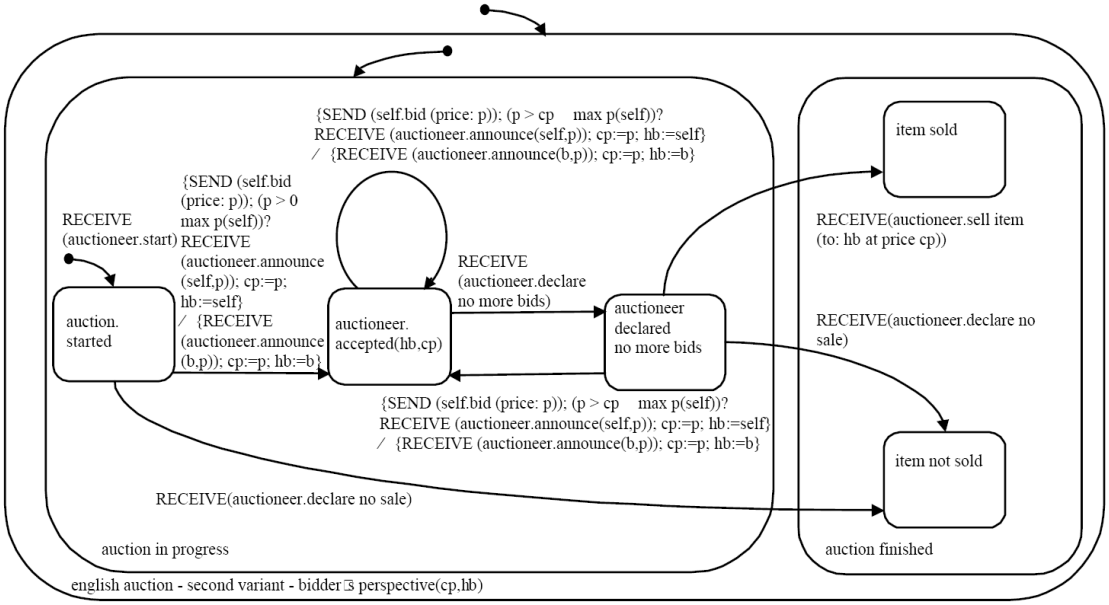


Figure 43. The protocol shown in Figure 42 from the point of view of a bidder (Dunn-Davies et al., 2005)

Formara and Colombetti (2003) propose a way to define interaction protocols using a commitment-based ACL. A commitment object consists of the following fields:

- a unique identifier
- a reference to the commitment's debtor
- a reference to the creditor
- the commitment's content, that is, the representation of the proposition to which the debtor is committed relative to the creditor
- a list of propositions that have to be satisfied in order for the commitment to become active;
- its state that can correspond to any one element of the finite set *{unset, cancelled, pending, active, fulfilled, violated}*
- a timeout valid only for *unset* commitments

A protocol is based on a set of speech acts as operations on commitment objects. It is described by an interaction diagram, that is, a graph whose nodes represent system states, and whose edges represent certain types of state transitions. In an interaction diagram, state transitions correspond either to speech acts performed by the interacting agents, or to environmental events strictly related to the interaction. The speech acts have a specific effect on commitments altering their state. Thus, in their work, Formara and Colombetti (2003) assume the existence of a specific mental model of the agents, the one related to commitments. Like in the previously presented work (Paurobally et al., 2004) they define both a logical and graphical method for representing the protocols.

König (2003) presents a new possibility in inter-agent protocols definition. He uses the state transition diagrams (STD) formalism to model protocols, but also decision activities, thus, using for both the same formalism. An STD is a special case of a Finite State Machine (FSM) that allows transitions between states either when an external or an internal event occurs to the system (according to his work, transitions in FSMs can only contain external events).

König defines a protocol as a structured exchange of messages. Then, he compares three approaches to modeling conversation policies, i.e. those based on STDs, FSMs and Petri nets. He observes that all approaches modeling conversations from the viewpoint of an observer are using either STD or petri nets, in contrast to those using FSM (or statecharts) that are representing the conversation from the viewpoint of a participating agent. For modeling a conversation from the point of view of a participating agent who receives and sends messages, König argues that a model supporting input and output operations is more suitable. When a conversation should be modeled from an observer's view, it is sufficient to use a model which is able to express that a message has been transmitted from one agent to another, like a transition in a STD or in a petri net. He chooses STD aiming to model both activities and protocols, allowing also for object-oriented development.

He makes the assumption that only two agents are involved in a protocol, i.e. the *primary* (who initiates the interaction) and the *secondary*. Moreover, the messages exchange is always synchronous, when one of them sends a message the other one is in a state of receiving a message (they cannot both be sending at the same time). Then he defines an FSM for the observer and from it he derives the FSMs of the participants. In a next level (higher level of abstraction) he defines communication acts that can make use of the protocols in the form of STDs. Finally, in a third level he defines the activities of the agents that can invoke one or more communication acts and assume a wait state until the acts finish. The acts themselves can choose to execute one or more protocols and enter a wait state until they are finished. All these can only happen sequentially.

Mazouzi et al. (2002), define protocols using the Colored Petri Nets (CPN) formalism. Petri Net emerged as a graphical tool for the description and analysis of concurrent processes which arise in systems with many components (distributed systems). A Petri net is a directed bipartite graph. It consists of places, transitions, and directed arcs. Arcs run between places and transitions, never between places or between

transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. Places may contain any non-negative number of tokens. A distribution of tokens over the places of a net is called a marking.

A transition of a Petri net may fire whenever there is a token at the end of all input arcs; when it fires, it consumes these tokens, and places tokens at the end of all output arcs. A firing is atomic, i.e., a single non-interruptible step. Execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire. If a transition is enabled, it may fire, but it doesn't have to. Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems. They were invented by Carl Adam Petri in 1939 (see the latest version of 2007).

Mazouzi et al. show how to transform an AUML interaction diagram to a CPN. They defined transformation templates, such as the one shown in Figure 44, for creating a petri net through an existing AUML AIP. In the figure the reader can see a protocol part where the initiator sends a request message, a query message or a not-understood message to the participant. This is transformed to a petri net by defining a transition with two outputs, one going to the next place of the initiator (the leftmost arrow in the petri net part of Figure 44) and the other to a place that is the input of three possible transitions, the “Send request”, “Send query” and “Send not-understood”. Only one of these will take the token to fire and produce its output place. The latter will be enabling a transition in the participant petri net.

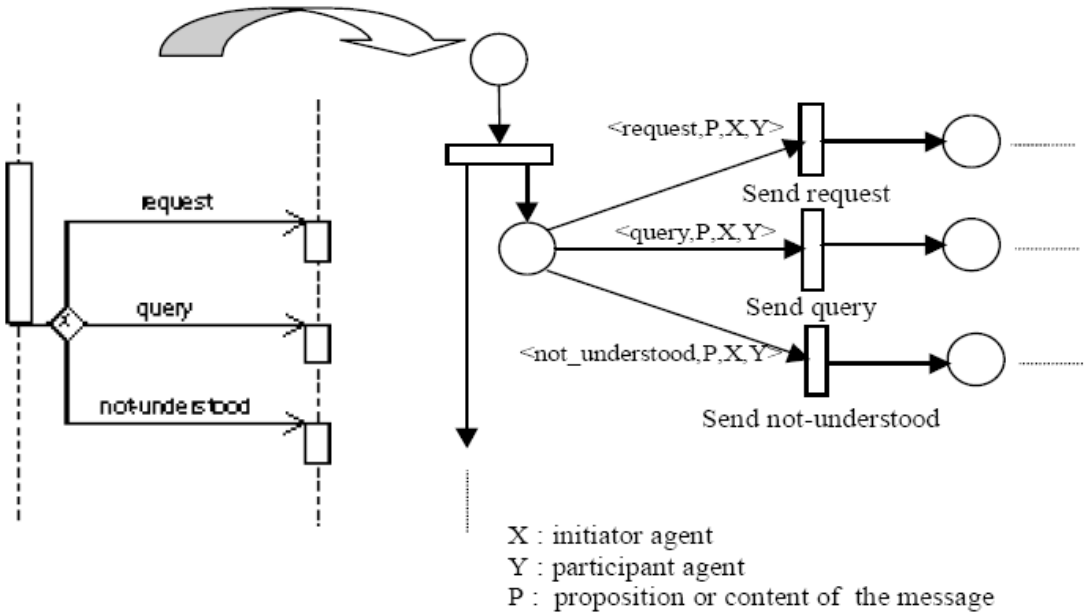


Figure 44. Transforming an exclusive OR part of an AUML AIP diagram to a CPN diagram part (Mazouzi et al., 2002).

Using such templates the AIP diagram in Figure 45 is transformed (or translated as the authors call this process) to the petri net in the same figure. See the application of the template of Figure 44 in the regions surrounded by dashed lines.

Their work allows for protocol reuse by defining ways to integrate existing protocols into new ones. Moreover, in their work the protocol complexity remains tractable (overcoming a major drawback of petri nets). However, CPN models in AOSE have yet to mature if they are to be used for creating agent models using an existing agent platform.

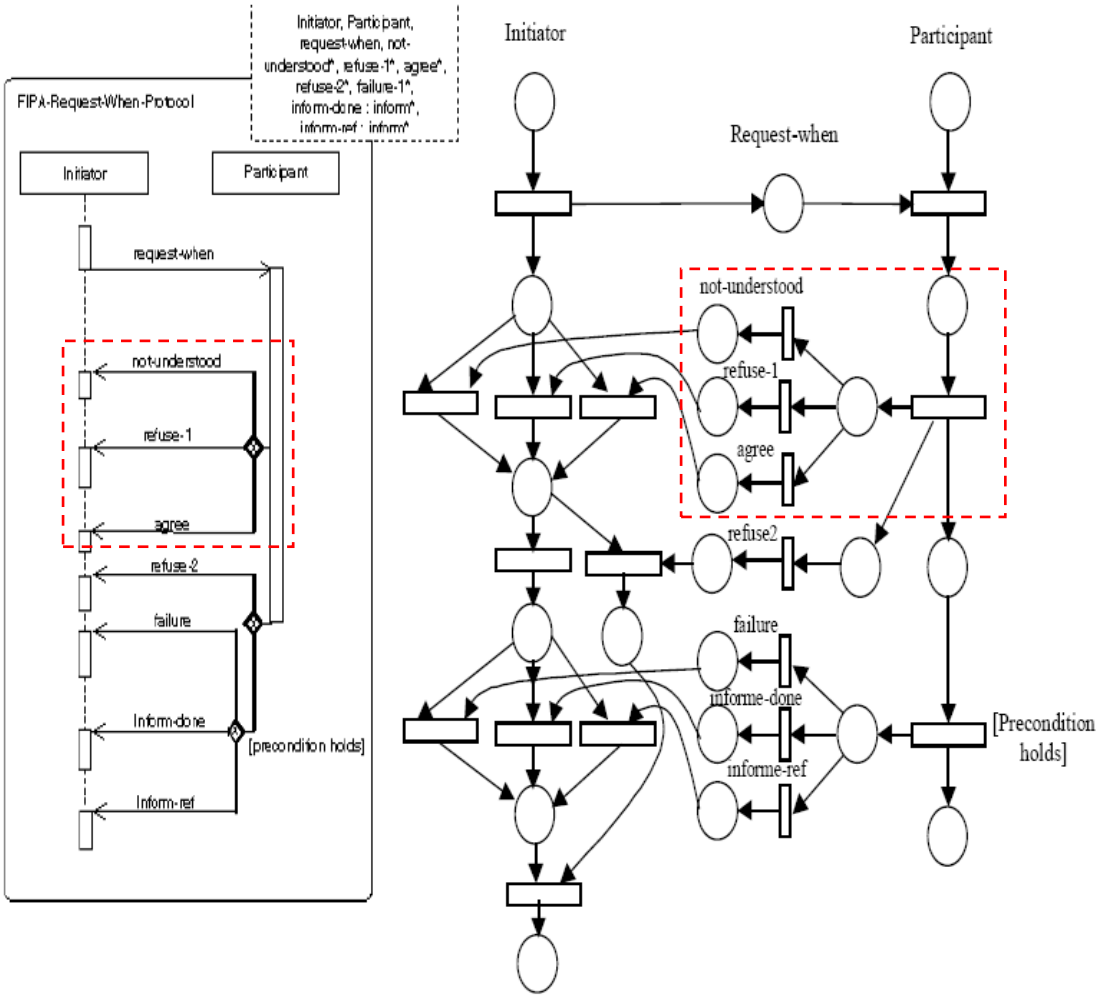


Figure 45. A translation of the FIPA-request-when protocol to a CPN (Mazouzi et al., 2002).

2.2.10 Model Driven Agents Development

The CAMLE (an acronym for the Caste-centric Agent-oriented Modeling Language and Environment) modeling language (Zhu and Shan, 2005) proposed a model driven

approach to the development of MAS leading to implementation using the language SLABS (an acronym for the Specification Language for Agent-Based Systems). CAMLE supports two software development phases, design and implementation. It proposes caste diagrams for defining the agent roles and their relationships. Collaboration diagrams define scenarios of the agents' interactions. In the agent level they define scenario diagrams and behavior diagrams. All these modes are CAMLE specific. CAMLE defines a transformation process for the behavior diagrams to SLABS code. Therefore, its applicability is limited as it is platform specific. Moreover, CAMLE does not cater for concurrency.

An interesting work is presented by Jayatilleke et al. (2005), where the authors propose a component based approach to designing Belief-Desire-Intentions (BDI) architectures. They define a general BDI framework that is expressed in XML format (PIM). Then they use an XSLT (XML transformation stylesheets) for defining the transformation of the XML model to JACK platform code (PSM model). Their work focuses in defining the XML metamodel for BDI-relevant entities as goals, events, triggers, plans, actions, beliefs, and, finally, agents and then in defining the XSL transformation to JACK code (Winikoff, 2005). However, the authors did not show how to define an XSLT for another platform.

Hahn et al. (2009) defined a metamodel (PIM4Agents) that can be used to model MAS in the Platform Independent Model (PIM) level of the Model-Driven Architecture (MDA). The added value of their work is that PIM4Agents instances can be instantiated with both the JADE and JACK agent development environments. Their approach is similar to the one followed in the Gaia2JADE process (as they state in their paper) for transforming roles to JADE behaviours. They define a metamodel for defining the behavior aspect of the PIM4Agents model. The *Behaviour* refers to a set of *Flows* that can be of type *InformationFlow* or *ControlFlow*. Each *Behaviour* contains a set of *Steps* that are linked to each other via a *Flow*. The *ControlFlow* describes in which order *Steps* are executed. The *InformationFlow* describes the order in which information flows between *Steps*. Each *Flow* connects exactly two *Steps*. A *Step* can be specialized as a *StructuredTask* or *Task*. A *StructuredTask* can be specialized to *Scope* and *Plan*. Both are connected to a *Condition* that mainly defines a set of facts that are connected by a logical operator. The *Plan* can have two *Conditions*, a precondition that has to be satisfied in order to execute the Plan and a post-condition that defines the fact that should be valid after the *Plan* execution.

2.2.11 Other works

Depke et al. (2002) introduced the idea of agents modeled as roles within the use cases diagram system box. In the requirements specification, analysis and design phases they worked with three different views of the system, a structural model, a functional model and a dynamic model. Each phase refines the models of the previous one. They change the semantics of UML class diagrams using them for the structural model by replacing class methods with messages and operations. For the dynamic model they use statecharts, however, they alter the way of forming

transition expressions allowing only the usage of class operations. For the functional model, they use attributed graphs in pairs showing the state of the system before and after the reception of an inter-agent message as graph transformation rules that define how an event changes the state of the system.

All in all, their approach is based on altered UML diagrams and attributed graphs presenting two disadvantages, a) they do not show how the final design models can be implemented (since there are no tools using their models), and b) the software engineers that are familiar with UML may be confused with the different and not adequately explained semantics. Moreover, the notion of inter-agent protocols is absent and their design does not address any known agent communication language.

2.2.11.1 The concepts of Capability and Functionality

The reason for exploring the existing uses of these terms is because they have been associated with the agent modeling process by many researchers and methodologies (e.g. the Prometheus methodology that has already been presented) and are also used within the methodology presented here-in. Therefore besides Prometheus the following interesting works are also discussed.

Braubach et al. (2005) proposed a capability concept for BDI agents. In their view, capability is *"a cluster of plans, beliefs, events and scoping rules over them"*. Capabilities can contain sub-capabilities and have at most one parent capability. Finally, the agent concept is defined as an extension of the capability concept aggregating capabilities. However, this capability concept is limited to the BDI agent architecture and in agent development an agent is something more than an assortment of capabilities. The agent should also be able to coordinate his capabilities.

Capability in AML (Trencansky and Cervenka, 2005) is used to model an abstraction of a behavior in terms of its inputs, outputs, pre-conditions, and post-conditions. A behavior is the software component and its capabilities are the signatures of the methods that the behavior realizes accompanied by pre-conditions for the execution of a method and post-conditions (what must hold after the method's execution). This approach is similar to service oriented architectures and, thus, considers the agent as an aggregation of services. Thus, in this case we have a simplistic definition of agent as an object that provides information about its methods similarly to SoA approaches.

2.2.11.2 Agile Agent Development

Knublauch's approach (2002) for extreme programming of MAS relies on process modeling to capture and clarify requirements, to visually document agent functionality, and to enable communication with domain experts. Their process metamodel was designed to be easy to comprehend and use by end users of the agent application, to be extensible for specific types of agents, and to allow for automatic and semi-automatic transformation into executable code. Thus process

models are deemed as very important for achieving an agile process. They use the AGIL-Shell for modeling the process using the Gaia models (mentioning that other tools, such as VISIO could also be used). Thus, they link the agile development process to using process modeling of MAS and their results provide evidence that an agile process such as XP is suitable for the development of MAS, even though their experiments were not using an agent platform and developed rather simple agents.

Chapter 3

The Agent Modeling Language (AMOLA)

The Agent MOdeling LAnguage (AMOLA) provides the syntax and semantics for creating models of multi-agent systems covering the analysis and design phases of the ASEME software development process. It supports a modular agent design approach and introduces the concepts of intra- and inter-agent control. The first defines the agent's lifecycle by coordinating the different modules that implement his capabilities, while the latter defines the protocols that govern the coordination of the society of the agents. The modeling of the intra and inter-agent control is based on statecharts. The analysis phase builds on the concepts of capability and functionality. AMOLA deals with both the individual and societal aspect of the agents.

3.1 The Basic Characteristics of AMOLA

The Agent Modeling Language (AMOLA) describes both an agent and a multi-agent system. Before presenting the language itself, some key concepts must be identified. Thus, the concept of *functionality* is defined to represent the thinking, thought and senses characteristics of an agent. Then, the concept of *capability* is defined as the ability to achieve specific goals (e.g. the goal to decide in which restaurant to have a diner this evening) that requires the use of one or more functionalities. Therefore, the agent is an entity with certain capabilities, including inter and intra-agent communication. Each of the capabilities requires certain functionalities and can be

defined separately from the other capabilities. The capabilities are the modules that are integrated using the *intra-agent control* concept to define an agent. Each agent is considered a part of a community of agents, i.e. a multi-agent system. Thus, the multi-agent system's modules are the agents and they are integrated into it using the *inter-agent control* concept.

The originality in this work is the intra-agent control concept that allows for the assembly of an agent by coordinating a set of modules, which are themselves implementations of capabilities that are based on functionalities. Here, the concepts of capability and functionality are distinct and complementary, in contrast to other works where they refer to the same thing but at different stages of development, e.g. in Prometheus (Padgham and Winikoff, 2005). The agent developer can use the same modules but different assembling strategies, proposing a different ordering of the modules execution producing in that way different profiles of an agent, like in the case of the KGP agent (see Bracciali et al., 2006). Using this approach, an agent can have a *decision making capability* that is based on an *argumentation based decision making functionality*. Another implementation of the same capability could be based on a different functionality, e.g. *multi-criteria decision making based functionality*.

Then, 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, for the latter see Pavón et al., 2005) or strongly imply (like Tropos) the agent mental models. Of course, there are some developers who want to have all these things ready for them, but there are others that 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.

The AMOLA models are related to the requirements analysis, analysis and design phases of the software development process. AMOLA aims to model the agent community by defining the protocols that govern agent interactions and each part of the community, the agent, focusing in defining the agent capabilities and the functionalities for achieving them. The details that instantiate the agent's functionalities are beyond the scope of AMOLA that has the assumption that they can be achieved using classical software engineering techniques. In the requirements analysis phase, AMOLA defines the *System Actors and Goals (SAG)* and the Requirements Per Goal (RPG) models. In the analysis phase AMOLA defines the *System Use Cases model (SUC)*, the *Agent Interaction Protocol model (AIP)*, the *System Roles Model (SRM)* and the *Functionality Table (FT)*. In the design phase AMOLA defines the *Inter-Agent Control (EAC) model* and the *Intra-Agent Control (IAC) model*.

Throughout this chapter, some parts of the analysis and design models of a real-world agent-based system, which was developed during this thesis, are presented. The requirements were to develop a system that allows a user to access a variety of

location-based services supported by a brokering system. The system should learn the habits of the user and support him while on the move. It should connect to an OSGi[2] service for getting the user's coordinates using a GPS device. It should also handle dangerous situations for the user by reading a heart rate sensor (again an OSGi service) and call for help. A non-functional requirement for the system is to execute on any mobile device with the OSGi service architecture. The broker has access to a variety of existing web services but should also provide added value services. For more details about the real-world system, which will be referred to as ASK-IT for the remainder of this document, the reader can refer to Moraitis and Spanoudakis, 2007.

3.2 The Requirements Analysis Phase Model

3.2.1 System Actors and Goals Model (SAG)

The AMOLA model for the requirements analysis phase is the *SAG* model that is composed by the Actor diagram, which is similar to the Tropos actor diagram (thus, a Tropos requirements analysis method fragment could be combined with minimal effort with ASEME), containing the actors and their goals. The SAG model is a graph involving actors who each have individual goals. A goal of one actor may be dependent for its realization to another actor; such a goal is also called *dependum*. The *depender* actor depends on the *dependee* in order to achieve the dependum. Graphically, actors are represented as circles and goals as rounded rectangles. Dependencies are navigable from the depender to the dependum and from the dependum to the dependee. Note that for simplicity of presentation, if a goal has no dependees is just drawn next to the depender. The goals are then related to functional and non-functional requirements in plain text form. An entity can qualify as an actor if it represents a real world entity (e.g. a "broker", the "director of the department", etc).

An example of a SAG model is presented in Figure 46. It is a subset of the SAG model for the ASK-IT System. This model was created after identifying the stakeholders relevant to this project (Spanoudakis et al., 2005). Such are the:

- *User*: The user is a mobility impaired person that wants to get infomobility services tailored to his needs (e.g. find the nearest toilet that is accessible according to his type of impairment). This user is assumed to wander in the environment having access to the internet and wherever possible access to local area networks using technologies like Wi-Fi. He also has constant access to devices and services that are on his person and move around with him.

[2] The Open Services Gateway initiative (OSGi) alliance is a worldwide consortium of technology innovators defining a component integration platform. Find out more in http://www.osgi.org

Such can be a GPS device. He also needs assistance in handling dangerous situations (e.g. if he has a heart attack)

- *Broker*: This is the ASK-IT B2C (Business to Consumer) Operator. He is interested in aggregating services offered by diverse service providers either globally or locally. Whenever a user makes a request he matches the request to his repository of available services and selects the most relevant one to request on behalf of the user.

- The Added Value Service Providers: These service providers can provide a simple service or they can introduce new added value services through the aggregation of one or more simple services accessed through the broker. A simple service provider offers map information for a specific city. An added value service provider offers map information for any city including the capability to add points of interest offered by many independent providers

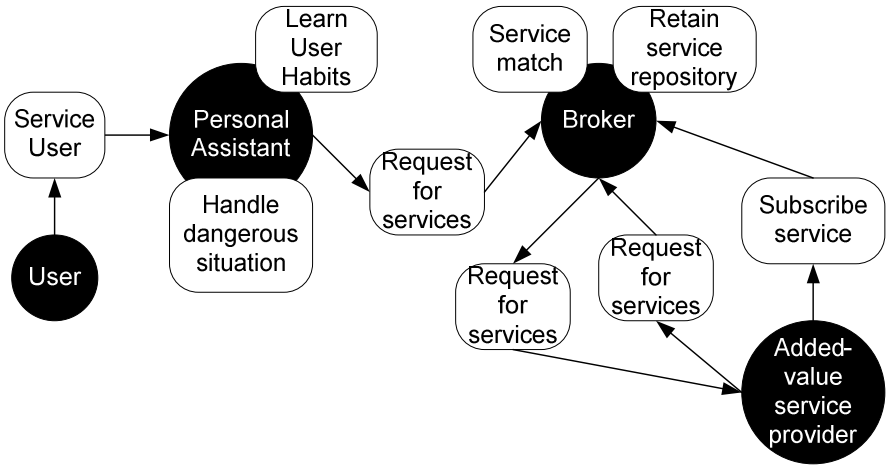


Figure 46. Actor diagram (or SAG model). The circles represent the identified actors and the rounded rectangles their goals.

The stakeholders are modeled as actors. A stakeholder that is assisted by software introduces a new actor, usually named as personal assistant. Thus, in Figure 46 the above three stakeholders are represented by four actors, the user, his personal assistant, the broker and the added value service provider. The user needs to get location based services and for that he is dependent to his personal digital assistant. The latter has three individual goals, to adequately service his user, to learn his/her habits and to autonomously handle a dangerous situation. The personal assistant depends on the broker (BR) for getting services. The broker represents a network operator or portal stakeholder who acts as a service aggregator and offers the services to its users. Its goals include the maintenance of a service repository, finding the best service for a user and accessing several web services offered by third parties. Moreover, he depends for getting added-value services to such a stakeholder (the “Added-value service provider” or AVSP), who provides specialized

services for users with special needs or capabilities. For example, an organization of mobility impaired persons maintains a repository of accessible streets and buildings and can provide trip planning services to such persons. For offering their service they depend on the broker themselves in order to get maps or public transport routing options.

3.2.2 The Requirements Per Goal Model

The Requirements per goal (RPG) is a simple model aiming to associate SAG goals to requirements presented in plain text mode. For adding the goal requirements the engineer should add the answers to the following questions:

- Why does the actor have this goal and why does he depend to another for it (this is the most important question and its answer is usually the goal's name)
- What is the outcome of achieving the goal (identify related resources)
- How is he expected to achieve this goal (identify the task to be performed for reaching this goal)
- When is this goal valid (identify timing requirements)

A non-functional requirement for the personal assistant's service user goal is to be executed on a mobile device. Another is that it should reply to a user request within 10 seconds (see Table 2).

Table 2. A portion of the Requirements Per Goal (RPG) model for the Personal Assistant Actor in ASK-IT project.

Personal Assistant goals	
Service User	Delivery of the service within 10 seconds
	The service is offered from a mobile device with the OSGi service architecture
	The user can request a mapping or a routing service

An implementation of AMOLA can choose to unify the two models (SAG and RPG) to one by adding a new property to the goal concept of the SAG model and catalogue the requirements related to that goal there (this is the approach followed in the AMOLA implementation in Chapter 5). As each requirement is related to a goal this is a logical approach. However, in the AMOLA specification these are left as two separate models for two reasons: The first is that by not altering the graphic representation of the Tropos actor diagram it is easy for Tropos practitioners to adapt to the AMOLA SAG model. The second reason is that a common practice in requirements management is to gather requirements in a tabular form, like the one

shown in Table 2, where they provide identification numbers to requirements for referring to them in the project lifeline.

3.3 The Analysis Phase Models

The main models associated with this phase are the *System Use Cases model (SUC)*, the *Agent Interaction Protocol model (AIP)*, the *System Roles Model (SRM)* and the *Functionality Table (FT).* The SUC is an extended UML *use case diagram* and the SRM is mainly inspired by the Gaia methodology (Wooldridge et al., 2000). Thus, a Gaia roles model method fragment can be used with minimal transformation effort.

3.3.1 The System Use Cases Model (SUC)

The use case diagram (SUC) helps to visualize the system including its interaction with external entities, be they humans or other systems. 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). The human roles are distinguished by their name that is written in italics. 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 general use cases can be decomposed to simpler ones using the *include* use case relationship. General use cases are also referred to as *capabilities*. A use case that connects two or more (agent) roles implies the definition of a special capability type: the participation of the agent in an *interaction protocol* (e.g. negotiation). A use case that connects a human and an artificial agent implies the need for defining a human-machine interface (HMI), another agent capability. A use case can *include* a second one showing that its successful completion requires that the second also takes place.

The SUC model presented in Figure 47 is part of the use cases for ASK-IT. Actually, it is a part focusing in the personal assistant (PA) role. The reader should notice at this point that the general use cases correspond to the goals of the requirements analysis phase. It is also important to note that at this phase the task of the system modeler is not to identify goals and dependencies between actors, like in the SAG, but to analyze the behavior of the system in order to achieve specific tasks. However, at the highest level of abstraction these tasks correspond to the system goals. The difference is that the know-how related to this phase is not that of the business modeler or the business consultant, it is that of the systems engineer or analyst.

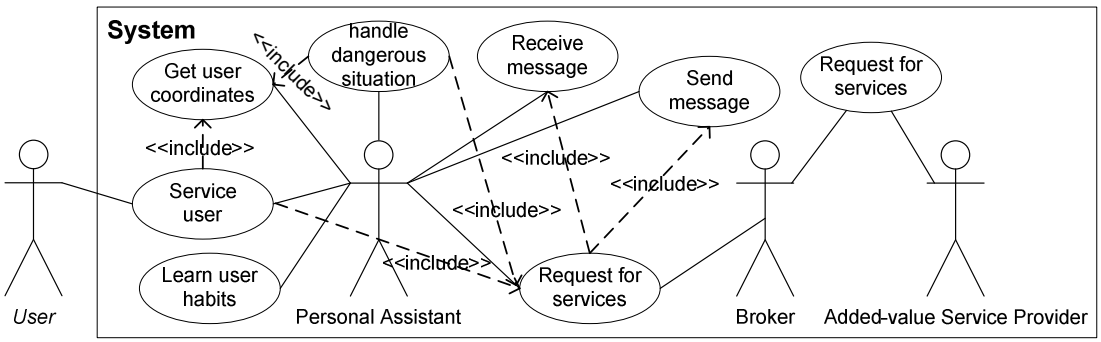


Figure 47. SUC Model: A Use Case diagram for the ASK-IT project.

3.3.2 The Agent Interaction Protocols Model (AIP)

An AIP (the reader should take care not to confuse it with the AIP model of AUML, for the remainder of this document AIP will refer to the AMOLA model) defines one or more participating agent roles, the rules for engaging (why would the roles participate in this protocol), the outcomes that they should expect in successful completion and the process that they would follow in the form of a *liveness* formula. The *liveness* formula is a process model that describes the dynamic behavior of the role inside the protocol. It connects all the role's activities using the Gaia operators (see Table 1). The *liveness* formula defines the dynamic aspect of the role, that is which activities execute sequentially, which concurrently and which are repeating.

As an example, the Request for Services AIP, which was built within the ASK-IT project, is presented in Table 3. This protocol is similar to the FIPA Request protocol (see FIPA TC Communication, 2002a) standard. There are two roles involved, the Service Requester (SR) and the Service Provider (SP). Someone would expect to see the personal assistant and the broker roles implicated, however, the reader should notice that the same use case exists between the broker and added-value service provider roles. Thus, the protocol is defined abstractly defining two abstract roles, the SR and SP. The rules for engaging and outcomes are described in free text format. However, the last part is where the process that needs to be followed by the participants is described in a liveness formula. The SUC model shows what a participant in a protocol does. At this point the question that needs to be answered is when the participant acts. So, by the SUC model the analyst can see that the SR sends and receives a message, however, in the AIP model he defines that he first sends the request message and then receives the response message.

This protocol is shared by many SUC roles (or *concrete roles*), for example the personal assistant (PA), the broker (BR) and the added-value service provider (AVSP) can use it as service requesters (SR). However, only the BR and the AVSP can use it as service providers (SP). The broker role is a classic broker as it has been defined by

Klucsh and Sycara (2001), i.e. the service requester knows how to form a valid request for processing by the service provider but he only interacts with the broker. Thus, the same protocol can be used both for the broker and the service provider.

Table 3. Agent Interaction Protocol for the ASK-IT system

Request for Services		
Participants	*Service Requester (SR)*	*Service Provider (SP)*
Rules for engaging	He needs to get an e-service within a specific amount of time	He will profit by providing a service within a specific amount of time
Outcomes	He has obtained the e-service results or a denial of service message or a service failure message, or no response	He has provided the e-service results or a denial of service message or a service failure message, or timed out
Process	request for services = send request message. receive response message	request for services = receive request message. process request. send response message

3.3.3 The Systems Roles Model (SRM)

The system roles model (*SRM*) is mainly inspired by the Gaia roles model (Wooldridge et al., 2000). A role model is defined for each agent role. The role model contains the following elements: a) the interaction protocols that this agent will be able to participate in, b) the liveness model that describes the role's behavior. The liveness model has a formula at the first line (*root formula*) where activities or capabilities can be added. A *capability* must be decomposed to *activities* in a following formula. The Gaia operators have been enriched with a new operator, the $|x^{\omega}|^{n}$, with which we can define an activity that can be concurrently instantiated and executed more than one times (n times).

The liveness formula grammar has not been defined formally in the literature, thus it is defined here using the Extended Backus–Naur Form (EBNF), which is a metasyntax (or metametamodel, as it was referred to in §2.1.4.4) notation used to express context-free grammars. It is a formal way to describe computer programming languages and other formal languages. It is an extension of the basic Backus–Naur Form (BNF) metasyntax notation. EBNF was originally developed by Niklaus Wirth (1996). The EBNF syntax for the *liveness* formula is presented in Listing 1, using the BNF style followed by Russel and Norvig (2003), i.e. terminal symbols are written in bold. The reader should note that the *process* property of the AIP model corresponds to the *formula* as it is defined in Listing 1.

A portion of the SRM for the personal assistant (PA), added-value service provider (AVSP) and broker (BR) roles in ASK-IT is presented in Figure 48. The PA role participates to the request for services protocol as the service requester. In his liveness model, the root formula states that he executes forever the "service user" capability in parallel with the "handle dangerous situation" capability. Each of these

capabilities is detailed in the following two formulas that have their name on the left hand side. Other compound elements are further detailed in following formulas.

Listing 1. The liveness formula grammar in EBNF format.

liveness	→ { formula }
formula	→ leftHandSide **=** expression
leftHandSide	→ string
expression	→ term
	| parallelExpression
	| orExpression
	| sequentialExpression
parallelExpression	→ term || term || … || term
orExpression	→ term | term | … | term
sequentialExpression	→ term **.** term **.** … **.** term
term	→ basicTerm
	| **(**expression**)**
	| **[**expression**]**
	| term*****
	| term**+**
	| term$^{\omega}$
	| |term$^{\omega}$|^{number}
basicTerm	→ string
number	→ digit | digit number
digit	→ **1** | **2** | **3** | …
string	→ letter | letter string
letter	→ **a** | **b** | **c** | …

The reader should note the interconnection between the role model (SRM) and the agent interaction protocol (AIP) model. For example, the Personal Assistant (PA) role in Figure 48, in the second line, indicates that he participates in the "Request for Services" protocol as a service requester (SR). This implies that the process part (from the AIP model in Table 3) related to an abstract protocol role (e.g. SR) that a concrete role (e.g. PA) assumes must be imported in the liveness model as-is. The

imported formulas in the liveness formulas of the three concrete roles shown in Figure 48 are written in italics.

The protocol participation related capability of a concrete role when the protocol has been defined for abstract roles includes the abstract role abbreviation so that the modeler can know which process field he must import. Therefore, if the PA role used in his liveness model the "request for services" capability, the modeler would not know whether he should import the SR or SP process of the protocol in the next formula. However, by using the name "request for services SR" (look at the last element of the right hand side of the third liveness formula of PA in Figure 48) the modeler knows that he should import the "send request message. receive response message" process from the AIP model in Table 3 (the part in italics in the next formula).

Role: Personal Assistant (PA)
Protocols: request for services: service requester
Liveness:
personal assistant = (service user)$^{\omega}$ || (handle dangerous situation)$^{\omega}$
service user = get user order. get user coordinates. get user preferences. request for services SR. present information to the user. learn user habits.
handle dangerous situation = invoke heart rate service. determine user condition. [get user coordinates. request for services SR]
request for services SR = search broker. [*send request message. receive response message*]
learn user habits = learn user preference. update user preferences.
Role: Broker (BR)
Protocols: request for services: service requester, request for services: service provider
Liveness:
broker = |request for services SP$^{\omega}$|^{10}
request for services SP = *receive request message. process request. send response message*
process request = service match. [(invoke data management | request for services SR)]
request for services SR = *send request message. receive response message*
Role: Complex Provider (CP)
Protocols: request for services: service requester, request for services: service provider
Liveness:
complex provider = |request for services SP$^{\omega}$|^{10}
request for services SP = *receive request message. process request. send response message*
process request = (decide route type. request for services SR. sort routes) | (decide POI types. request for services SR. decide POIs. request for services SR)
request for services SR = *send request message. receive response message*

Figure 48. A portion of the SRM model for three roles of the ASK-IT project

The analyst can then choose to add activities to the protocol part in the liveness formula but he has to keep the imported part intact. For example, the analyst has added to the PA another activity at the right hand side of the “request for services” formula, i.e. the “search broker” (see the liveness formulas of the PA role in Figure 48) and makes the execution of the protocol optional (putting all the protocol’s activities inside brackets). However, the imported protocol process part “send request message. receive response message” remains unchanged.

There is a reason for this restriction when building the liveness formulas. An agent must comply with a protocol specification. This means that he must do his part of the protocol as it is specified. The agent is free to do things before or after a protocol is executed. The agent is also free to determine how to achieve the tasks specified by the protocol (but he must do them and in the specified order).

Generalizing, at the liveness formulas part of SRM the analyst can add as many formulas as he sees fit. Here the role is analyzed in a top-down process. Another example is the use of the Service Provider (SP) process part of the “Request for Services” protocol (see Table 3) by the broker role (in Figure 48). The analyst has chosen to expand the “process request” activity to more specific ones in a next formula having the “process request” on the left hand side.

3.3.4 The Functionality Table (FT)

The functionality table is where the analyst associates each activity participating in the liveness formulas of the SRM to the technology or tool (functionality) that it will use (see an example of FT in Figure 49 for the capabilities of the PA). The *communicate* capability includes the “send message” and “receive message” activities and is shared by all agents as proposed by the Foundation of Intelligent Physical Agents (FIPA). This is the point where the analyst proposes the use of a platform for instantiation, e.g., in our example, the Java Agent Development Environment (JADE), an open source framework that adheres to the FIPA standards. This strategic choice also defines the programming language that will be used, in this case Java.

Returning to the ASK-IT example, the non-functional requirement for the PA to execute on any mobile device running OSGi services reveals that such a device must at least support the Java Mobile Information Device Profile (MIDP), which offers a specific record for storing data. Therefore, the activities that want to store or read data from a file must use the MIDP record technology.

An AMOLA implementation might choose to incorporate the functionality table in the SRM model adding a “functionality” property to each activity (as is done in Chapter 5). This approach is more compact and would better serve the needs of the developer. However, a decision maker would prefer to see this information in a tabular format (like in Figure 49) in order to gain a quick understanding about the technologies involved in developing each agent.

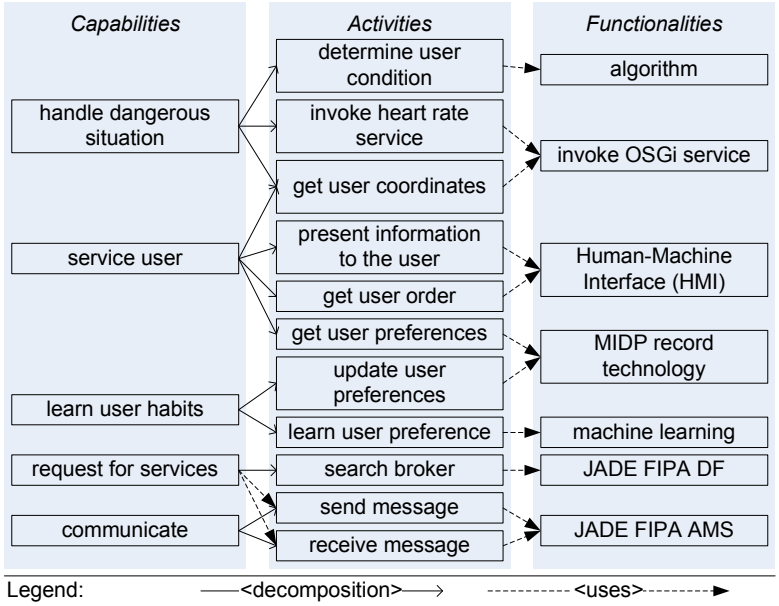


Figure 49. Functionality Table for the personal assistant role of the ASK-IT project.

3.4 The Design phase Models

The models associated with the *Design phase* are the *inter-agent control* and *intra-agent control*. They define the functional and behavioral aspects of the multi-agent system. In the past, the MaSE methodology (Deloach et al., 2001) defined agent behavior as a set of concurrent tasks, each specifying a single thread of control that integrates inter-agent as well as intra-agent interactions. The AMOLA models go one step further by modeling the interaction among the capabilities of an agent, i.e. what they call the different threads of control, but also their execution cycle. The model associated to the first level of this phase is the inter-agent control, which defines interaction protocols by defining the necessary roles and the interaction among them. The implementation of the inter-agent control is done at the agent level via the capabilities and their appropriate interaction defined via the intra-agent control. Finally, in the third level each capability is defined with regard to its functionality, what technology is used, how it is parameterized, what data structures and algorithms should be implemented. The intra-agent control model (IAC) defined in this phase corresponds to the Platform Independent Model (PIM) level of MDA.

3.4.1 The Inter-Agent Control Model (EAC)

The *inter-agent control* is defined as a statechart. It should be initialized by transforming the *agent interaction protocols* of the analysis phase to statecharts. Harel and Kugler (2004) present the statechart language adequately but not formally. Several authors have presented formal models for this language, as such an approach is needed for developing relevant statecharts-based CASE tools. David et al. (2003) proposed a formal model for the RHAPSODY tool, Mikk et al. (1997) for the STATEMATE tool and Wąsowski and Sestoft (2002) for VisualSTATE. The first one has been used as basis for the definition of the AMOLA statecharts as it is the first intended for object-oriented language implementation (VisualSTATE and STATEMATE are for C language development). These models not only formally describe the elements of the statechart, they also focus on the execution semantics. Mikk et al. (1997), for example adopt the Z language for modeling statecharts. However, this issue is out of the scope of this work. It is assumed that, as long as the language of statecharts is not altered, a statechart can be executed with any semantics available depending on the available CASE tool. The formal model that is adopted here-in is a subset of the ones presented in the literature as there are several features of the statecharts not used herein, such as the history states (which are also defined differently in these works).

Before formally defining the statechart for the EAC model, the elements that compose the transition expressions are examined. Then, the transition expressions are defined in EBNF. Transitions are usually triggered by events. Such events can be:

a) a sent or received (or perceived, in general) inter-agent message,

b) a change in one of the executing state's variables (also referred to as an intra-agent message),

c) a timeout, and,

d) the ending of the executing state activity.

The latter case is also true for a transition with no expression. Note that each state automatically starts its activity on entrance. A message event is expressed by *P(x,y,c)* where *P* is the performative, *x* is the sender role, *y* the receiver role and *c* the message body.

The items that the designer can use for defining the state transition expressions are the message performatives, the ontology used for defining the messages content and the timers. An agent can define timers as normal variables initializing them to a value representing the number of milliseconds until they timeout (at which time their value is equal to zero). The transition expressions can use the *timeout* unary predicate, which is evaluated to true if the timer value is equal to zero, and false otherwise. Timers are initialized in the action part of a transition expression, while the *timeout* predicate can be used in both the event and condition parts of the transition expression depending on the needs of the designer.

Besides inter-agent messages and timers there is another kind of events, the *intra-agent messages*. The change of a value of a variable can have consequences in the execution of a protocol. The variables taking part in a transition expression imply the fact that they are defined in the closest common ancestor OR state of the source and target states of the transition or higher in the statechart nodes hierarchy. Listing 2 shows the grammar for defining the transition expressions of the statechart. The intention regarding the performative definition is not to enumerate all possible performatives, the modeler can define such as he sees fit.

Listing 2. The statecharts transition expression grammar in EBNF format.

transitionExpression	→ [event] [**[**condition**]**] [/action]
event	→ **timeout(** variable **)**
	| performative**(**variable, variable, variable**)**
	| variable (comparisonOperator | equalNotEqualOperator) (variable | value)
	| predicate ∧ variable (comparisonOperator | equalNotEqualOperator) (variable | value)
	| quantifier index, (performative**(**$variable_{index}$, variable, variable**)** [∨ performative**(**$variable_{index}$, variable, variable**)**] | performative**(**variable, $variable_{index}$, variable**)** [∨ performative**(**variable, $variable_{index}$, variable**)**]), (cardinalValue | intNumber) comparisonOperator index comparisonOperator (cardinalValue | intNumber)
	| event logicalOperator event
	| **(**event**)**
quantifier	→ ∃ | **∀**
index	→ letter
cardinalValue	→ |set|
set	→ string
condition	→ variable (comparisonOperator | equalNotEqualOperator) (variable | value)
	| condition logicalOperator condition
	| **(**condition**)**
	| **not(**condition**)**
	| predicate **=** (**True** | **False**)
action	→ variable **=** (variable | value)
	| action connectiveOperator action

	| predicate **=** (**True** | **False**)
comparisonOperator	→ > | ≥ | < | ≤
equalNotEqualOperator	→ **=** | **≠**
connectiveOperator	→ **;**
arithmeticOperator	→ **+** | **−** | **/** | *****
logicalOperator	→ ∧ | ∨
variable	→ string
predicate	→ string **(**string [term] **)**
term	→ **,** (string | value) | term**,** (string | value)
value	→ intNumber | realNumber | "string" | ∅
realNumber	→ intNumber**.**intNumber
intNumber	→ digit | digit intNumber
digit	→ **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **0**
string	→ letter | letter string
letter	→ **a** | **b** | **c** | …
performative	→ **request** | **inform** | **propose** | …

Each state is defined by its name, the variables that are used by its activity and transition expressions, its type and the algorithm that is executed as its activity. Its type can be one of OR, AND, CONDITION, BASIC and two specializations of the basic state, START and END. Start is represented by a black dot and END by a black dot in a circle. Some definitions of the used background concepts (e.g. graphs and trees) will be required and, thus, are given before the formal statechart's definition.

Definition 3.1 (Knuth, 1997)**.** A *graph* is generally defined to be a set of points (called *vertices*) together with a set of lines (called *edges*) joining certain pairs of distinct vertices. There is at most one edge joining any pair of vertices. Two vertices are called adjacent if there is an edge joining them. If V and V' are vertices and if $n > 0$, we say that ($V_0, V_1,..., V_n$) is a *path* of length n from V to V' if $V = V_0$, V_k is adjacent to V_{k+1} for $0 \leq k < n$, and $V_n = V'$. The path is *simple* if $V_0, V_1,..., V_{n-1}$ are distinct and if $V_1,..., V_{n-1}, V_n$ are distinct.

Definition 3.2 (Knuth, 1997)**.** A graph is *connected* if there is a path between any two vertices of the graph. A *cycle* is a simple path of length three or more from a vertex to itself.

Definition 3.3 (Rosen, 1999)**.** A *directed graph (V, E)* consists of a set of vertices V and a set of edges E that are ordered pairs of elements of V (Rosen, 1999).

Definition 3.4 (Rosen, 1999)**.** A *tree* is defined to be a connected graph with no cycles (Rosen, 1999).

Theorem 3.1 (Rosen, 1999)**.** An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices

Definition 3.5 (Rosen, 1999)**.** A tree a particular vertex of which is designated as the root is called a *rooted tree*.

Since there is a unique path from the root to each vertex of the graph (from Theorem 3.1) each edge is directed away from the root. Suppose that *T* is a rooted tree. If *v* is a vertex other than the root, the *parent* of *v* is the unique vertex *u* such that there is a directed edge from *u* to *v*. When *u* is the parent of *v*, *v* is called a *child* of *u*. A vertex of the tree is called a *leaf* if it has no children. Vertices that have children are called *internal vertices*.

Definition 3.6 (Rosen, 1999)**.** An *ordered rooted tree* is a rooted tree where the children of each internal vertex are ordered.

To produce a total order of the vertices of an ordered rooted tree all the vertices must be labeled. This is achieved recursively as follows:

1. Label the root with the integer 0. Then label its *k* children (at level 1) from left to right with 0.1, 0.2, 0.3, …, 0.*k*.

2. For each vertex *v* at level *n* with label *A*, label its k_v children, as they are drawn from left to right, with *A.1*, *A.2*, …, $A.k_v$.

Thus, A.1 means that A is the parent of A.1.

The definition below for the statechart is inspired by the definition proposed by David et al. (2003).

Definition 3.7. A *statechart* is a tuple (*L, δ*) where:

- *L* = (*S, λ, Var, Name, Activity*) is an ordered rooted tree structure representing the states of the statechart.
    - $S \subseteq \mathbb{N}^*$ is the set of all nodes in the tree.
    - λ: $S \rightarrow$ {AND, OR, BASIC, START, END, CONDITION} is a mapping from the set of nodes to labels giving the type of each node. For $l \in S$ let *AND(l)* denote that $\lambda(l)$=AND. Similarly *OR(l)* denotes that $\lambda(l)$=OR and the same holds for all labels. START and END denote those nodes without activity, which exist so that execution can start and end inside OR-states. BASIC corresponds to a basic state. A condition state is denoted as CONDITION. START, END, BASIC and CONDITION nodes are leaves of *L*.
    - *Var* is a mapping from nodes to sets of variables. *var(l)* stands for the subset of local variables of a particular node *l*.

- *Name* is a mapping from nodes to their names. *name(l)* stands for the name of a particular node *l*.
- *Activity* is a mapping from nodes to their algorithms in text format implementing the processes of the respective states. *activity(l)* stands for the algorithm of a particular state that is represented by node *l*.

- $\delta \subseteq S \times TE \times S$ is the set of state transitions, where *TE* is a set of *transitionExpression* elements (see Listing 2).

The following are also defined according to the definitions of David et al. (2003):

Definition 3.8. Let *l* an internal vertex of an ordered rooted tree *L*. We call *sons(l)* = $\{l.x \in S | x \in \mathbb{N}\}$ the children of *l*

Definition 3.9. Let *l, k* two vertices of an ordered rooted tree *L* such that $\exists x \in \mathbb{N}$, $k.x = l$. Then the vertex *k* is called parent to *l* and it is denoted as *parent(l)*

Definition 3.9. Let *l* a vertex of an ordered rooted tree *L*. Then, the ancestors of *l* are defined as $ancestors(l) = parent(l) \cup ancestors(parent(l))$

A state name that starts with the string "send" implies an inter-agent message sending behavior for the activity of the state. A send state has only one exiting transition and its event describes the message(s) sent. Similarly, a state name that starts with the string "receive" implies that the activity of the state should wait for the receipt of one or more inter-agent messages. Again, the type and number of the expected messages is implied by the events monitored by the transition expressions that have this state as source.

This formalism allows for environment-based communication by defining state activities that monitor for a specific effect in the environment. This effect can be expected to be caused by any other agent or a particular agent. Such activities can be, for example, "wait for someone to appear" or "wait until my counterpart lifts the object" respectively.

The ontology can be defined in object-oriented format or in logic based format. The definition of the ontology using one format does not forbid the development of the system using the other, as, for example, Spanoudakis and Moraitis (2008) defined a way to encode an ontology that was developed using the Protégé ontology editor, in object-oriented format, to Prolog (logic programming) format (see the details in §4.9.3).

In Figure 50 a portion of the ASK-IT ontology (a complete report can be found in Spanoudakis and Moraitis, 2006b) is presented showing the concepts:

- CallParameter. This concept describes a service parameter. Such a parameter has a type (withType property), and value (different value properties one for each parameter type supported, for example, if the type is string the property withStringValue will store the value).

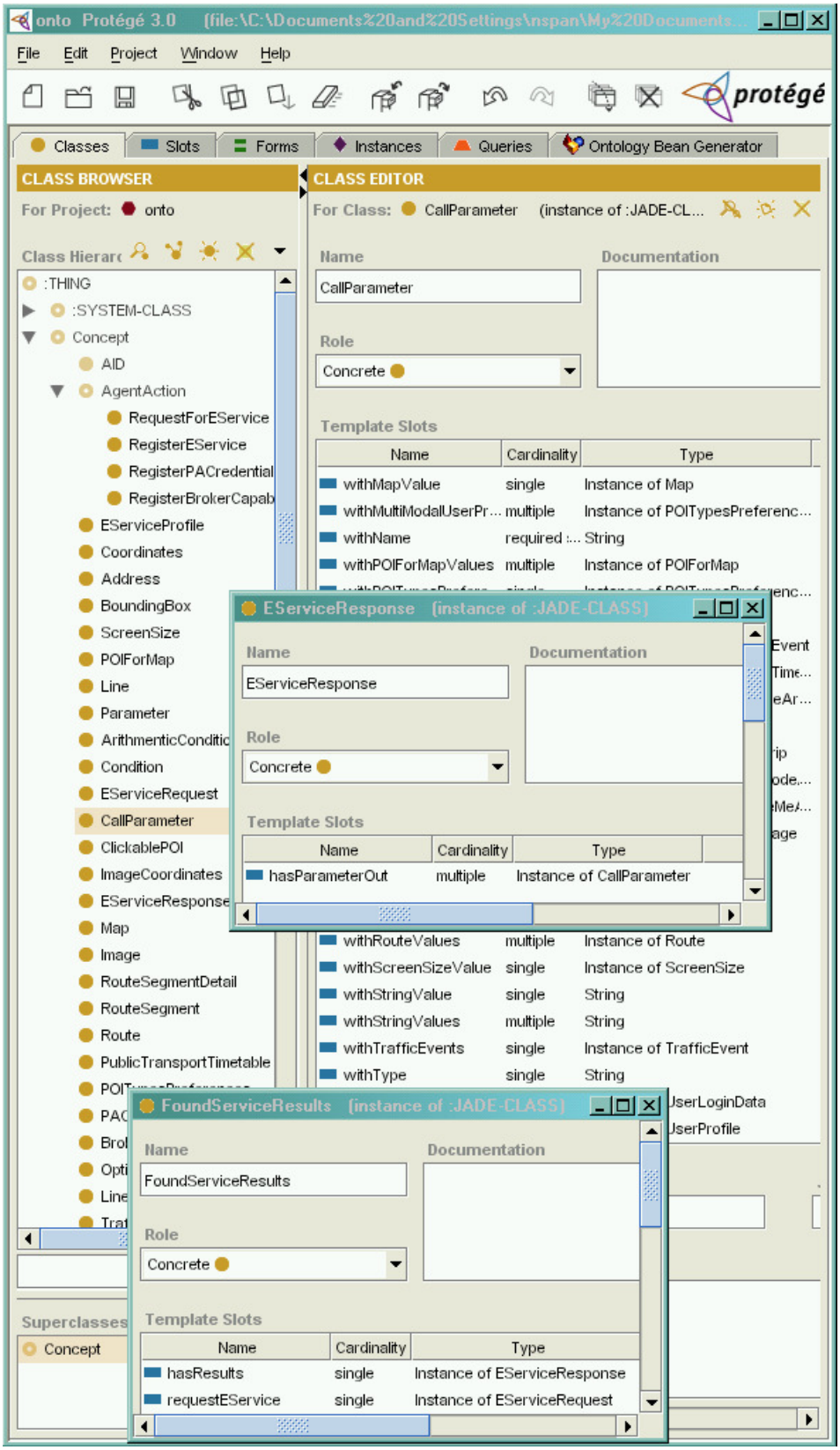


Figure 50. A snapshot of the ASK-IT ontology. The concepts *FoundServiceResults*, *EServiceResponse* and *CallParameter* (Spanoudakis and Moraitis, 2006b).

- EServiceResponse. An instance of this concept has one property with *multiple* cardinality. This means that its hasParameterOut property is a list of instances of type CallParameter.

- FoundServiceResults. An instance of this concept is returned as the content of the *inform* message of the service provider (SP) role, if, of course, the service invocation has been successful. Its properties are:

  a) an instance of the EServiceRequest concept, which is the content of the original service requester (SR) role's *request* message,

  b) an instance of the EServiceResponse concept.

Having formally defined the statechart as it is used in AMOLA it is now possible to proceed to the definition of the inter-agent control (EAC) model. The EAC is a statechart that contains an initial (START) state, an AND-state named after the protocol and a final (END) state. The AND-state contains as many OR-states as the protocol roles named after the roles. Two transitions connect the START state to the AND state and the AND state to the END state.

The EAC for the ASK-IT "Request for Service" protocol is presented as an example in Figure 51. The two participating roles each have their own orthogonal component. The service provider (SP) is waiting for a message request. The service requester (SR) starts by sending a message request, then both execute a transition, the SR to the "receive response message" activity and the SP to the "process request" activity. As soon as the message processing ends, the SP replies to the SR and both complete the protocol. In the case of a timeout the protocol again ends for both roles. Note the START node and END nodes of the protocol as a black dot and as a black dot inside a circle at the left side of the "Request for Services" AND state.

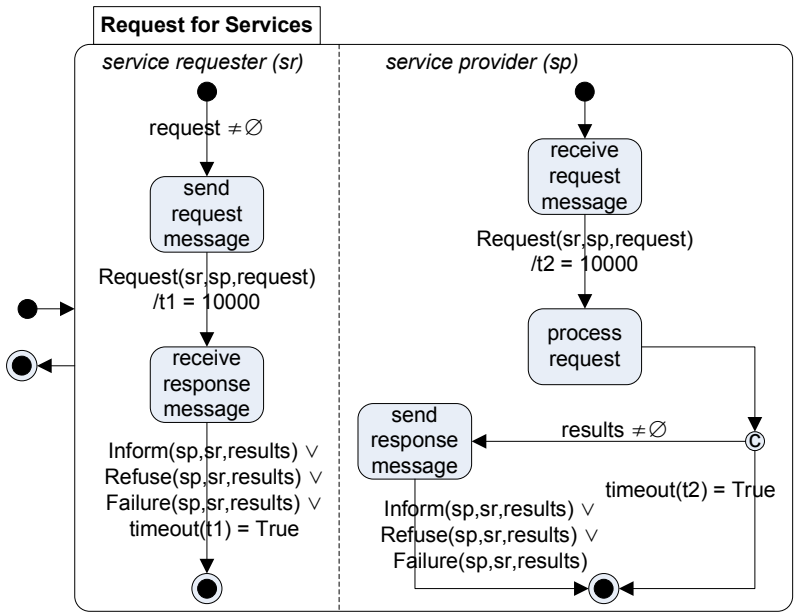


Figure 51. The EAC model for the Request for Services Protocol of ASK-IT project.

For the service requester (SR, the left hand side OR state inside the AND state in Figure 51) the protocol starts as soon as the request variable is assigned a value. This is denoted by the request ≠ ∅ intra-agent event. Then the SR goes to the "send request message state". It remains in that state until the message to its counterpart, i.e. the service provider (SR), is sent. The event of sending this message, which must have the *Request* performative, the SR as sender, the SP as receiver and the message content being the value of the *request* variable as the Request(sr, sp, request) atom suggests, causes another transition, targeting the "receive response message" state. The same transition executes the action t1 = 10000 that sets the timer *t1* to timeout in 10000 milliseconds (or 10 seconds). The receipt of an inter-agent message with performative *Inform*, *Refuse*, or *Failure* from the SP with a value for the *results* variable in its content, or the timeout of the timer *t1* can cause the final transition of this OR state to its END state.

Let's now see what the service provider (SP, the OR state inside the AND state in Figure 51 on the right) does all this time. The SP is automatically transited to the "receive request message" state (as the transition from the START state has no expression). There, the SP executes an activity that waits for a message by a SR. The SP transition to the next state has the same expression with the SR transition from the "send request message state" to the "receive response message". Therefore, when he gets to the next state, i.e. the "process request", the SP has received the *Request* message from the SR and he has set his timer (*t2*) to timeout in 10000 milliseconds. Now, the SP has two possibilities for making a transition. The first possibility is that he assigns a value to his *results* variable, while the second is that the *t2* timer timeouts. In the first case there is a transition enabled targeting the "send response message" state, while in the second case there is a transition to the END state that finishes the protocol execution for this role. From the "send response message" state the SP can take the transition to the END state after sending an *Inform*, *Refuse*, or *Failure* message to his counterpart.

Here the reader should note that the inter-agent control model does not impose a specific way for interpreting the exchanged messages or a technology for exchanging them. These issues are defined by the developers according to the platform that they will use for deploying their system and their expertise. For example, in FLBC (Moore and Kimbrough, 1995) the effects of a *request* message are linked to the beliefs of the sender which may not be the case in another communication language with different semantics. Thus, a procedural agent might not have a model of beliefs in contrast with a BDI (i.e. belief-desire-intention, see Dastani et al., 2005) agent.

At this point, the main difference with Moore's proposal (2000) becomes evident. In the EAC model, all states represent activities, while in Moore's work they just represent a point in time where a condition is true (like in finite state machines).

The EAC model defines an inter-agent protocol addressing the disadvantages of statecharts as they were presented by Paurobally et al. (2003) and discussed in §2.2.9.3. Participating roles are shown explicitly, each having his own orthogonal component in the statechart. The issue of compound transitions is handled by not using them for building the statechart. The question of completeness is handled by

introducing the obligatory association of a state to an activity. In the case of protocols, the participants that are not the initiators ensure the fact that a sub-state will be entered by having a first state that waits to receive a message.

3.4.2 The Intra-Agent Control Model (IAC)

In the agent level, the intra-agent control (IAC) is defined using statecharts in the same way with the inter-agent control model (EAC). The difference is that the top level state (root) corresponds to the modeled agent (which is named after the agent type). One IAC is defined for each agent type.

In Figure 52 the IAC model for the personal assistant (PA) agent in the ASK-IT project is presented, while in Figure 53 the reader can see the IAC model for the broker (BR) agent. In the next chapter an algorithm for automatically generating these models from the SRM will be presented. The reader can see that the IAC model of the PA uses the request for services protocol as a service requester (SR) two times. The broker implements the service provider (SP) role of the same protocol and within the "process request" state he uses the request for services protocol as a SR. This is how the IAC model can integrate protocols with other agent capabilities seamlessly.

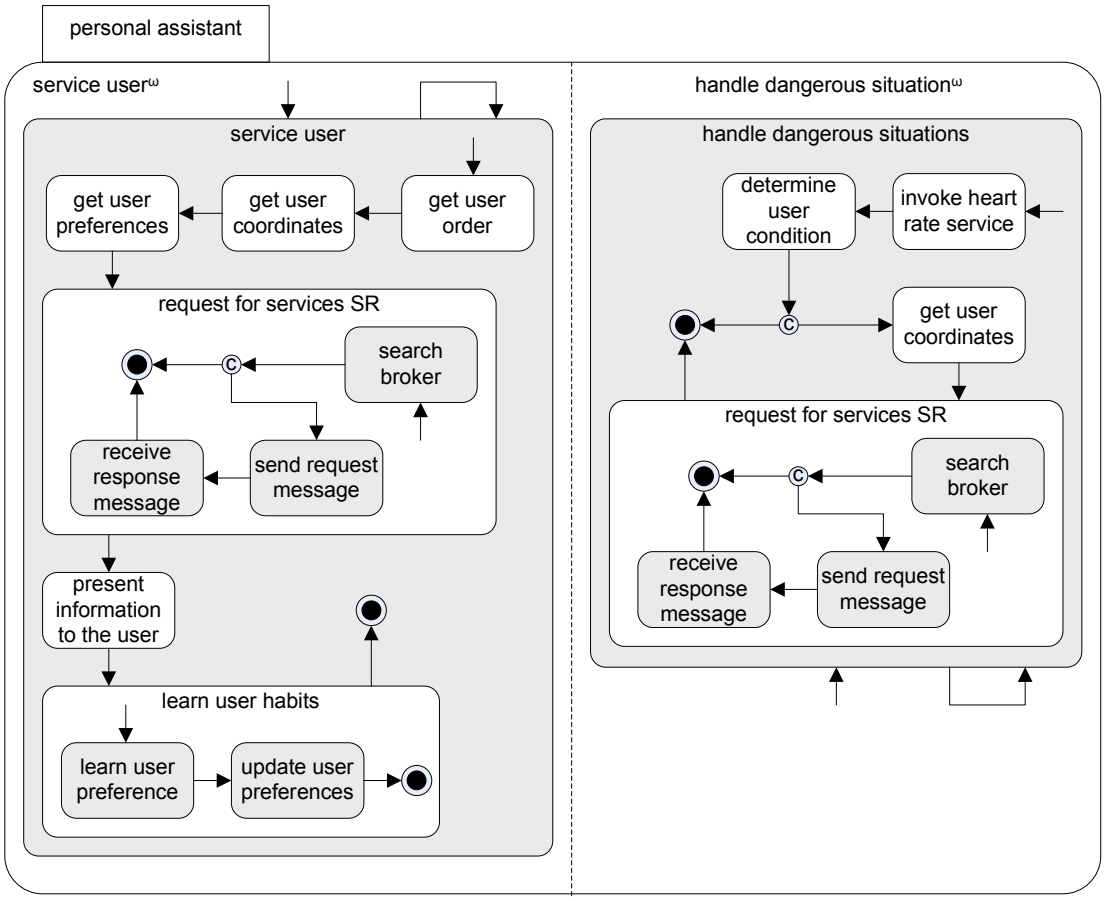


Figure 52. IAC model for the Personal Assistant agent in the ASK-IT project.

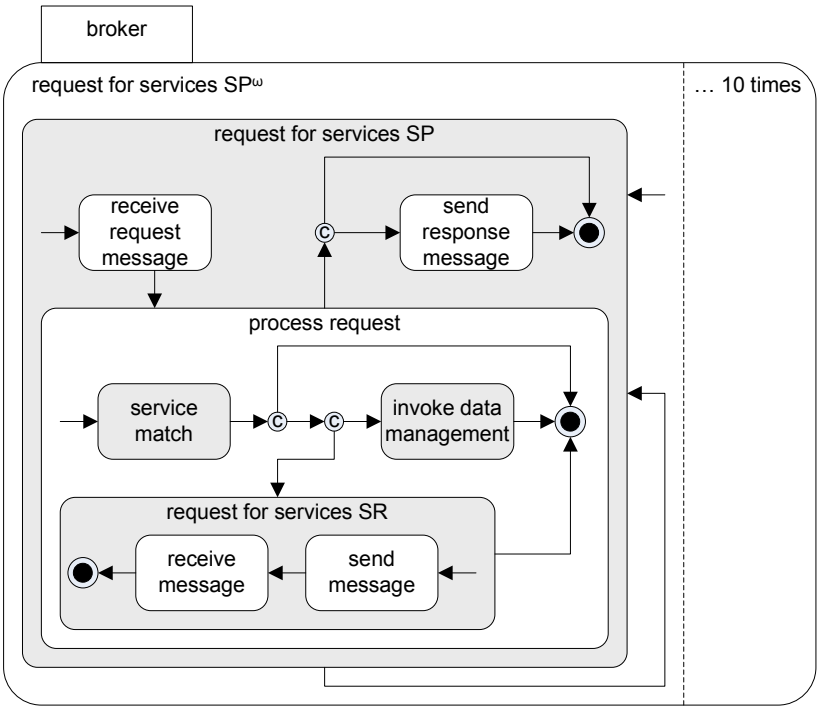


Figure 53. IAC model for the Broker agent in the ASK-IT project.

The IAC model can accommodate the use of diverse agent architectures. For example, Figure 54 shows how we can model an existing BDI architecture (i.e. the one proposed by Dastani et al., 2005) using AMOLA. The lifecycle of this agent starts in the "receive message" state. Then, as soon as a message arrives or another enabling event occurs the BDI agent enters the "apply goal planning rules" OR state. Within that OR state, more specific activities match the goals with rules, select rules matching the agent's beliefs and apply a goal planning rule. The next OR state, i.e. "apply plan revision rules" and its substates find rules matching to the plans, select rules matching the agent's beliefs and apply the selected plan revision rule. Finally, the agent reaches the "execute plan" state that depending on the selected plan may send a message, take an external action or an internal (or mental) action, or do nothing. After finishing the plan execution the agent returns to his message receiving state. This is an example of how someone can use the IAC model to coordinate the agent's capabilities and to accommodate a well-known type of architecture in a platform independent manner, i.e. the way to implement this model is not yet chosen at this time.

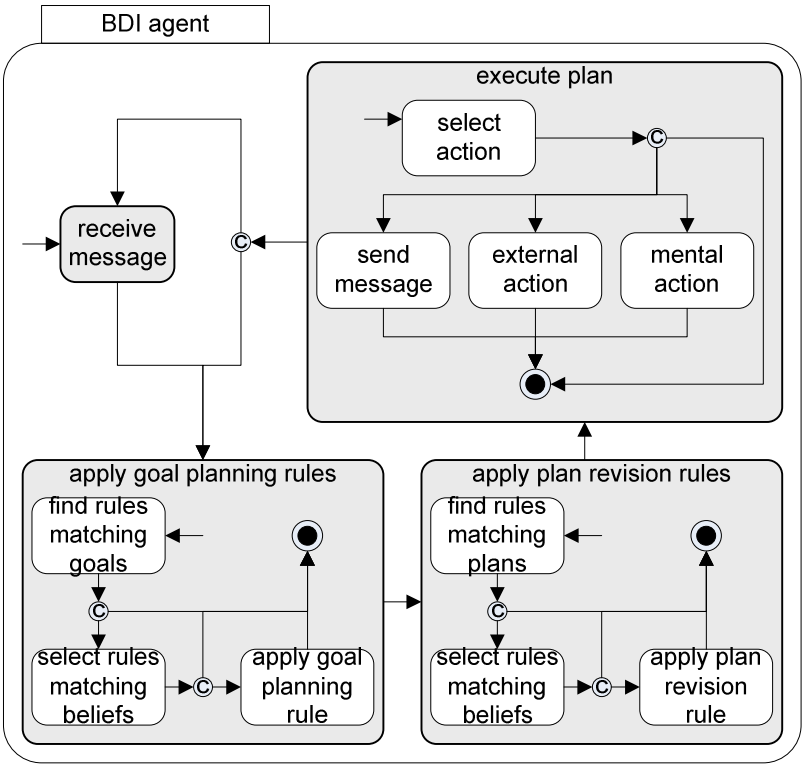


Figure 54. The intra-agent control model of a BDI agent

Chapter 4

The Agent Systems Engineering Methodology (ASEME) Process

The goal of this chapter is to define a methodology and process for agent-based systems development. Such a process starts from system requirements, continues with analysis, design, implementation and testing ever since the waterfall model. The modern processes add detail on how to program (e.g. pair programming in XP), how to manage a project, when the client has to participate, etc. These issues are out of the scope of this work.

This work aims to deliver a lightweight software development process. Lightweight means that it will limit the engineering artifacts down to the necessary ones that will allow for the development of small to large scale systems and the application of all existing approaches to a development process, i.e. waterfall or iterative development. The process will demonstrate the steps needed for development. The complexity will be within the steps, thus an ontology for a simple project can be one or two classes, while for a complex project it can range to hundreds of classes.

The reasons why there is still room for a new methodology in AOSE and what are the challenges related to the proposal of a new one such as ASEME are presented before venturing into the description of the ASEME process

4.1 Why a New Methodology in AOSE

AOSE has been established as a research field the last years. A large number of methodologies and works in this field have contributed to the state of the art. However, there is still room for a new methodology that will try to integrate important missing characteristics. The latter along with the reasons for selecting them are discussed in this paragraph.

First of all, there should be a *straightforward transformation process between the models of the development phases* following the guidelines of the modern MDE paradigm. MDE addresses three non-functional requirements in software engineering, i.e. portability, interoperability and reusability. Moreover, a clear transformation process between the software development phases from requirements analysis down to implementation allows for traceability of requirements. This allows for a developer to answer the question of which part of his program addresses which requirement. A number of authors in AOSE have introduced concepts and ideas from the model-driven engineering domain. Most of them just introduce a 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 by Perini and Susi (2006) and from a BDI (Belief-Desire-Intention) representation in XML format to JACK platform code by Jayatilleke et al. (2004). A more recent work García-Magariño et al. (2009) 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. However, until today, no methodology provides an MDE approach defining model transformations between all the software development phases.

Considering agents as complex software there should be *different levels of abstraction* that will allow system modelers to have a clear view of the whole system and be able to "zoom in" different components and view more detail. This approach has been considered as very important even from the era of Data Flow Diagrams. The right level of such abstractions would include:

- a society level where the modeler can have a static and a dynamic view of the agents and their interactions
- an agent level where the modeler can inspect the parts/modules (or components of the agent) and the behavior of an individual agent
- the capability level, where the modeler can inspect the details of each of the agent's modules.

The first two levels are directly related to AOSE, the capability level is more related to classic software engineering. The different software development phases should each define its models regarding all these three levels of abstraction. Existing methodologies usually start by defining the agents and their protocols in an analysis (or architectural design) phase and then focus in each agent in a design (or detailed design) phase.

The models of an agent design language should be *agent mental model independent* as there is yet no globally recognized formalism for defining it. Thus the developer should be able to define any kind of agent architecture he desires. However, it would be very useful to define templates or models adhering to the most important agent architectures. Thus, the methodology should include two useful abstractions identified in the literature of AOSE, namely *capability and functionality*. These have been used with different meanings in the past in order to provide new concepts for modeling agent-based systems with relation to previous methodologies like, e.g. for object-oriented development.

Another important issue in a new methodology should be the definition of a proper *integration method between inter-agent protocols and agent capabilities*. Existing methodologies achieve this through a step that is not automated and depends on the engineer's capability to mentally transform a type of diagram (e.g. an AUML Agent Interaction Protocol) to another (e.g. an activity diagram).

We believe that the *module based approach* proposes the right level of decomposition of an agent because it allows for the reusability of the modules as independent software components in different types of agents, having some common capabilities. Thus the new methodology should allow for modular design and modules reuse.

A new methodology should also provide guidelines as to *what constitutes an agent*. In most of the methodologies agents are just aggregates of capabilities without explaining what is the right level of complexity of an agent, when does the system modeler identify an agent entity. Sometimes, the agent methodologies forget the issue of agenthood and when an identified software component should qualify as an agent.

Finally, the works in the AOSE domain do not adequately address the issue of *non-functional requirements*. Only Tropos provides a means for documenting them in the requirements analysis phase as soft goals. Then, Tropos uses them in two ways. The first is to evaluate identified tasks as helping or restricting the soft-goals. The second is to identify tasks that pursue the soft-goals (in which case soft goals become functional requirements). Non-functional requirements need a way to influence the way to implement a system or task (see Garcia et al., 2006) and this is what a new methodology must seek.

4.2 ASEME Process Overview

ASEME uses AMOLA and defines the software development process using the Software Process Engineering Metamodel (SPEM) language. SPEM allows for the graphical representation of software development processes using the notation presented in Figure 55. A *Process* is defined as a series of *Phases* that produce *Work Products*. In each phase simple or complex *activities* take place. A complex activity is

defined as a *Work Definition*. Activities are achieved by *Human Roles* and they may produce *Intermediate Products* which can be either *Text* or *Graphical Models*. The phases are described by *Process Packages* that include the associated roles and work products. Each process package defines a process that contains work definitions and /or activities connected through dashed arrows like in flowcharts. The black dot shows where the process starts and the black dot in a circle where it ends (like in statecharts). An activity (or work definition) has input and output products. An arrow from an activity to a product means that the product is created (or updated) by the activity. An arrow from a product to an activity means that the product is an input of the activity.

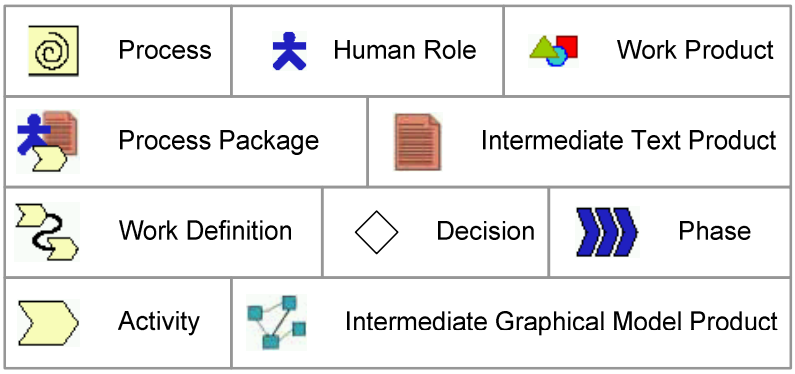


Figure 55. The Software Process Engineering Metamodel (SPEM) Notation.

The next paragraph provides an overview of the process while the following ones provide details for each development phase. A small example on how to develop a meetings management system is used throughout this section for the ASEME process demonstration. This example (the meetings management system) has been widely used in the past for demonstrating the use of methodologies (e.g. for the Prometheus and MAS-CommonKADS methodologies in Henderson-Sellers and Giorgini, 2005). This system's requirements are, in brief, to support the meetings arrangement process. The user needs to be assisted in managing his meetings by a personal assistant. The latter manages the user's schedule and services the user. The meetings organization process is managed by the secretariat to which the users submit their requests to schedule a new meeting or change the date of an existing one. The secretariat contacts the users' assistants whenever she needs to negotiate a meeting date.

The software development phases of the Agent Systems Engineering Methodology (ASEME) are presented in Figure 56. There are six phases, the first four produce system models (development phases), while the last two (verification and optimization phases) evaluate and optimize these models.

The ASEME process is iterative, allowing for incremental development and provides the original possibility to jump backwards to any previous phase due to the utilized model driven engineering (MDE) approach. MDE allows this possibility as the models

of a subsequent phase are created by those in the previous phase. As this process is straightforward the inverse is also possible (return to the model of a previous phase using the model of a next phase). This is also called *reverse engineering*. For example, having a Java class a developer can easily reverse engineer the class diagram (see Figure 57). This is because the process of transforming a class diagram to code is automatic and straightforward.

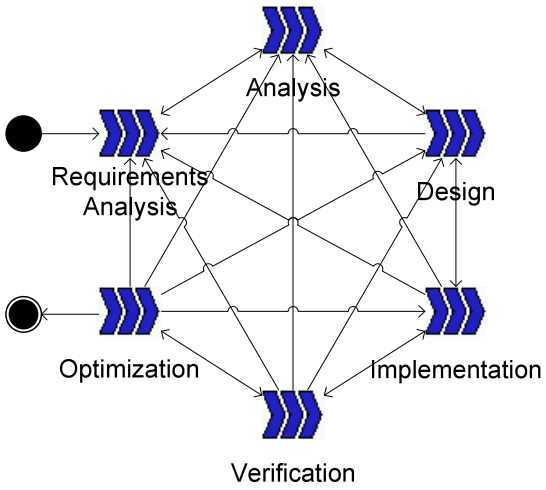


Figure 56: ASEME Process Overview.

In ASEME the SAG, SRM, IAC and a Platform Specific Model (PSM) are the main models outputted by the requirements analysis, analysis, design and implementation phases respectively. Each of these models is produced by applying simple transformation rules to the previous phase model and this transformation is traceable, that is it can be reverse engineered.

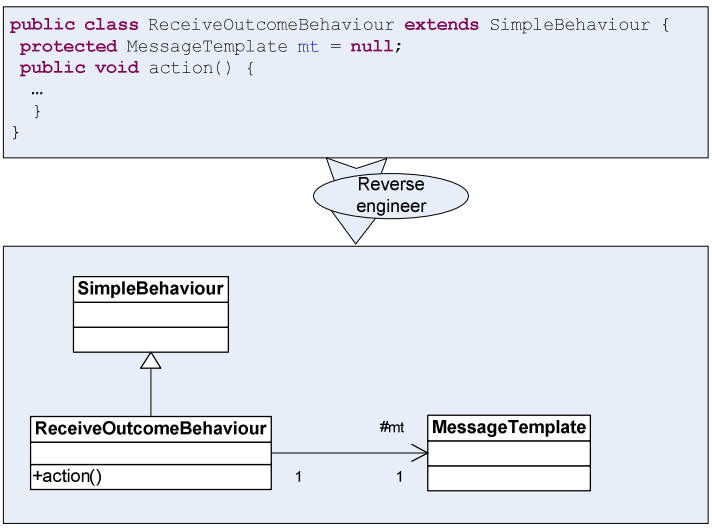


Figure 57. Reverse Engineering from Java code to a Class diagram.

Three levels of abstraction are defined for each phase. The first is the *societal level*. There, the whole multi-agent system functionality is modeled. Then, the *agent level* zooms in each part of the society, i.e. the agent. Finally, the details that compose each of the agent's parts are defined in the *capability level*. The reader is reminded that the concept of *capability* is defined as the ability of an agent to achieve specific tasks that require the use of one or more *functionalities*. The latter refer to the technical solution(s) to a given class of tasks. Moreover, capabilities are decomposed to simple *activities*, each of which corresponds to exactly one functionality. Thus, an activity corresponds to the instantiation of a specific technique for dealing with a particular task. ASEME is mainly concerned with the first two abstraction levels assuming that development in the capability level can be achieved using classical (or even technology-specific) software engineering techniques.

In Figure 58, the ASEME phases, the different levels of abstraction and the models related to each one of them are presented. The phases, the human roles participating in each of them and their products are presented using the SPEM notation in Figure 59. Some of the products (like the ontology product of the design phase) are not AMOLA models and in the next paragraph their type and purpose will be explained. In the same figure the reader can see the human roles that are expected to work at each phase. In the following each of these phases will be enriched with a process definition.

Development Phase	***Levels of Abstraction***		
	Society Level	***Agent Level***	***Capability Level***
Requirements Analysis *AMOLA Models*	Actors `Actor Diagram`	Goals `Actor Diagram`	Requirements `Requirements per goal`
Analysis *AMOLA Models*	Roles and Protocols `Use case Diagram, Agent Interaction Protocols`	Capabilities `Use case Diagram, Roles Model`	Functionalities `Functionality Table`
Design *AMOLA Models*	Society Control `Inter-agent control model, Ontology, Message Types`	Agent Control `Intra-agent control model`	Components
Implementation	Platform management code	Agent code	Capabilities code
Verification	Protocols testing	Agent testing	Component testing
Optimization	Number of instantiated agents	Agent resources	Code optimization

Figure 58: ASEME phases and their AMOLA products.

An important characteristic of ASEME is automation. A large number of activities are automated, which means that the modeler has nothing more to do than execute a simple command, or “click” in the used Integrated Development Environment (IDE). These activities are those that transform a model to another model. In the SPEM process diagrams such activities are named after the pattern <MODEL A>2<MODEL B>. Chapter 5 describes the technical solutions for automating these activities using the Eclipse IDE. Thus, the ASEME developers would benefit very much from using eclipse and the projects presented in Chapter 5. However, the ASEME process describes in detail the transformation procedure therefore even a modeler that uses a blackboard for drawing his models can follow it.

The Eclipse Foundation (created in 2004) is a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate both an open source community and an ecosystem of complementary products and services (for more information visit http://www.eclipse.org/).

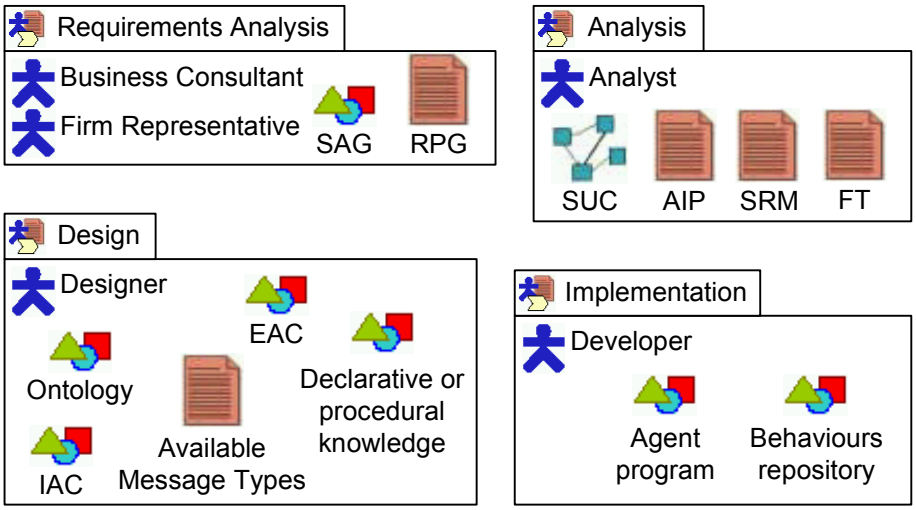


Figure 59. ASEME process packages.

4.3 Requirements Analysis Phase

In Figure 60, the requirements analysis phase is presented in SPEM notation. The three levels of abstraction are represented by the three activities. In the society level, the actors and their goals that depend on other actors are defined; in the agent level, the individual goals of each actor are identified, and, in the capability level, specific requirements, functional and non-functional, are assigned to each one of these goals. The output of the requirements analysis phase is the SAG model,

containing the actors and their goals and the RPG (Requirements per Goal) table that associates requirements to each goal. All these activities are manual, usually performed by a business consultant (a representative of the organization that will develop the software) together with a firm representative that is the client.

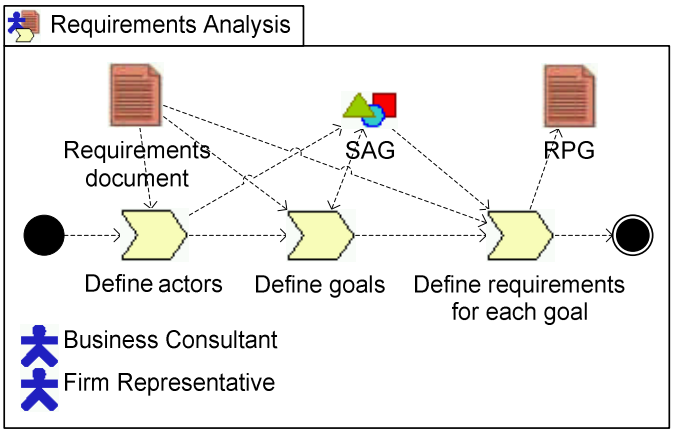


Figure 60: The ASEME Requirements Analysis Phase.

Regarding the running example for the meetings management system, the actors involved are the user and the assistant that helps him to manage his meetings. The latter is adaptable to a specific user, thus is modeled as a personal assistant. Moreover, there is the secretariat role that is represented by the meetings manager actor. The reader can see the SAG model in Figure 61. The goal of the user to manage his meetings is dependent on the personal assistant. In the agent level individual goals are defined; one such is the adaptation to user needs for the personal assistant, named “learn user habits”. In the capability level the functional and non-functional requirements for each goal are defined in free text.

Eclipse supports the modeler in every activity by providing graphical or text editors for the AMOLA models. These editors are based in the AMOLA metamodels presented in §5.1. The *Sample Reflective Ecore Model Editor* of Eclipse is shown in Figure 62. In this figure the business consultant edits the SAG model (it is the same one shown in Figure 61). The reader can see the properties of the selected goal (LearnUserHabits) at the bottom of the figure (note that the goal requirements are included as one of the goal’s properties). The reader can also notice that the edited file (SAGModel.xmi) has the *xmi* extension which implies that it is a standardized XML file (this file is included in Annex 6). Thus, an experienced consultant in XML can edit this file directly using a text editor. The task of improving the AMOLA graphical editors is part of the future work and it is intended that in the future the editor will have the same look as the SAG model in Figure 61.

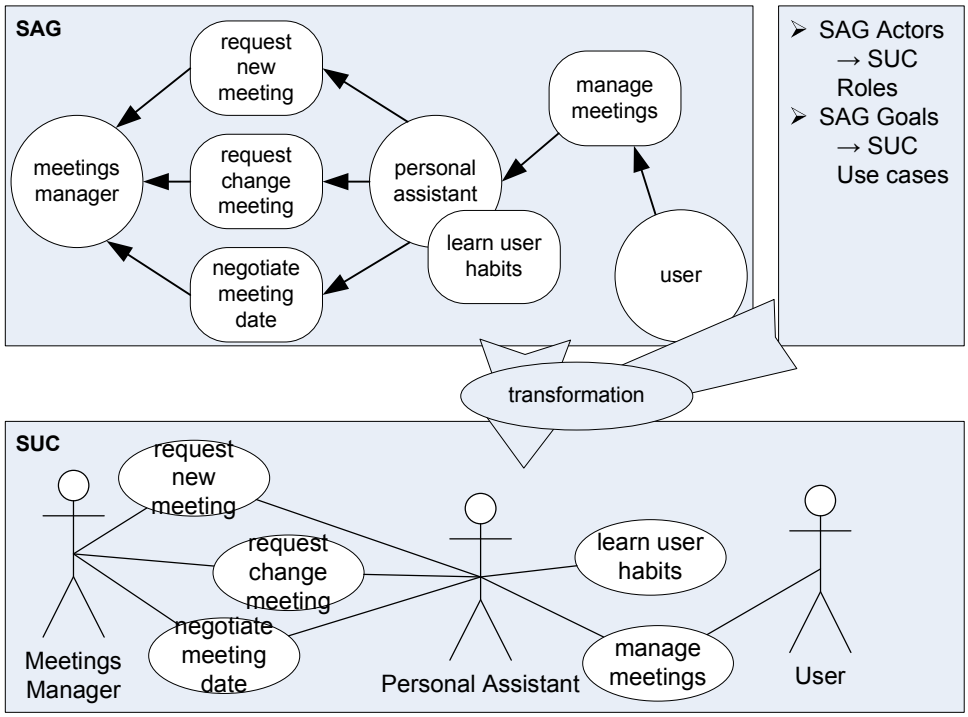


Figure 61: The SAG2SUC transformation.

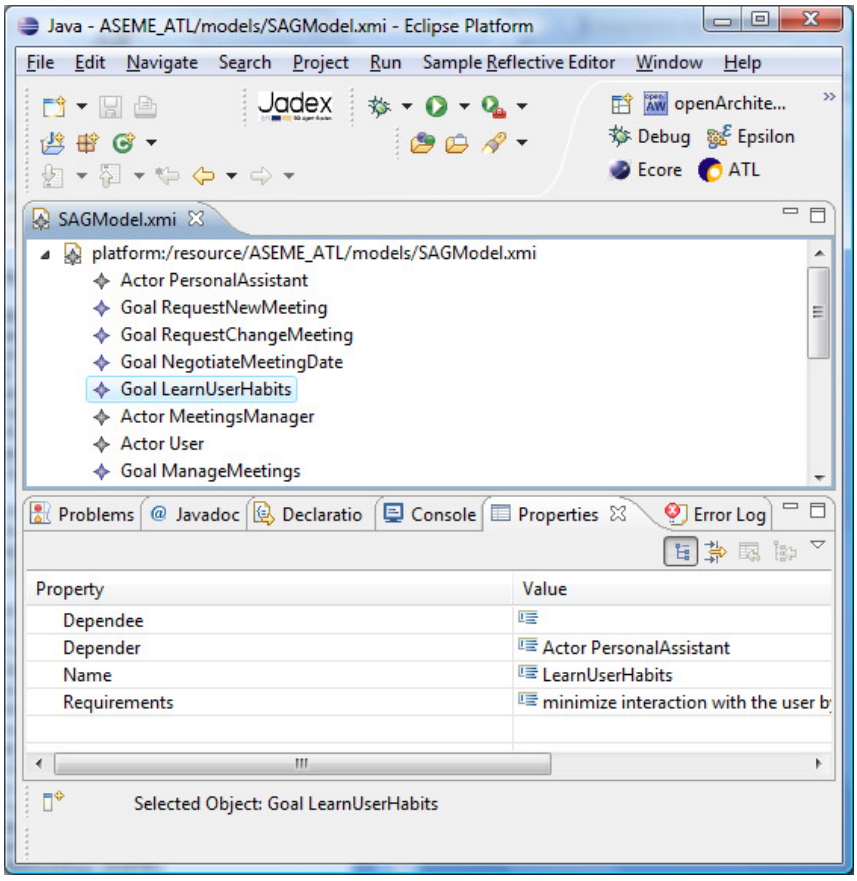


Figure 62. A graphical editor of the Eclipse IDE.

4.4 Analysis Phase

The ASEME analysis phase is presented in Figure 63. The first activity transforms the SAG model to an AMOLA use case diagram (SUC model). The model transformation process (SAG2SUC) is straightforward and is graphically presented in Figure 61. The actors are transformed to roles and the goals to use cases. The SUC model is used as an intermediate model, used only within the analysis phase, aiming to facilitate the creation of the SRM model, which is the output of the analysis phase. This activity can be automated if the analyst uses the Eclipse IDE with the SAG2SUC project presented in §5.2.1.1.

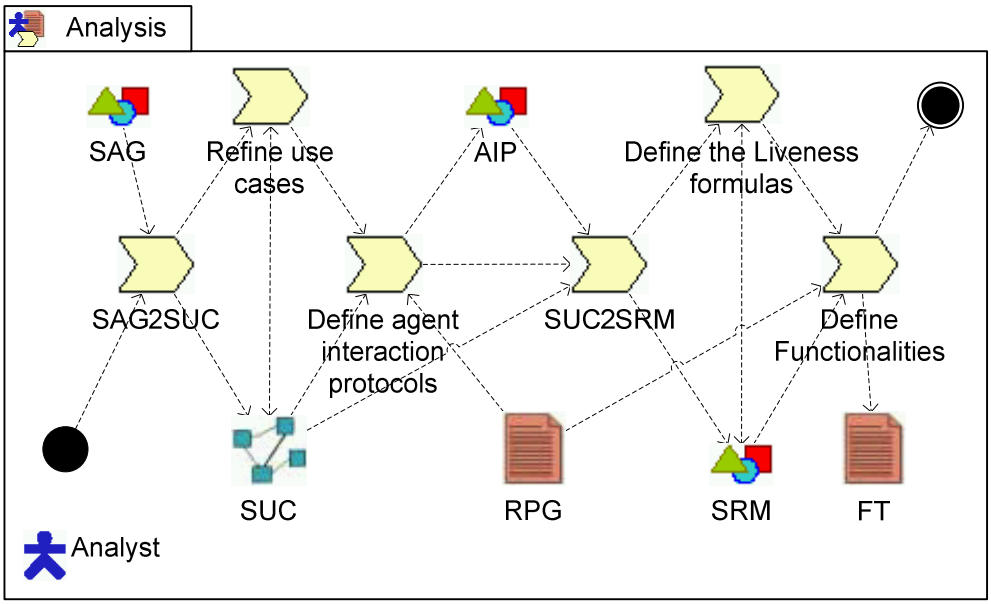


Figure 63: The ASEME Analysis Phase.

The next activity refines the use cases in order to produce a rich SUC model. In the society level, the use cases connecting two agent roles are elaborated. Concurrently, in the agent level, the use cases that have only one role participant are refined. UML suggests the use of sequence diagrams and activity diagrams for refining use cases. Another possible approach is the use of the MAS-CommonKADS task model (Iglesias, 1998) method fragment or research efforts in Goal-Oriented Requirements Engineering (Lamsweerde, 2001). Note that the use of such methods is optional, a modeler can work directly on the use case diagram. The analyst refines the "request new meeting" use case for the "personal assistant role" and assigns cardinality to the "negotiate meeting date" use case showing that two or more "personal assistant" roles are involved (see the SUC model in Figure 64). The *Sample Reflective Ecore Model Editor* of Eclipse or any XML editor can be used for graphically or textually (respectively) editing the SUC model.

It is important to note that during the SUC refinement activity, the analyst no longer refines goals (like in the SAG model), but he tries to analyze the behavior of the system to specific tasks. The difference is that the know-how related to this phase is not that of the business modeler or the business consultant, it is that of the systems engineer.

The next ASEME activity is about defining the *Agent Interaction Protocols* (AIP), or simply protocols. Protocols (in the society level) originate from use cases that connect two artificial agent roles (e.g. the "request new meeting" is a protocol use case). For example, in Figure 61, there is the "request new meeting" use case that was created by transforming a goal of the requirements analysis phase. Let's call such use cases (those that correspond to a goal) the *general use cases*. In the SUC model in Figure 64 the "request new meeting" is refined for the "Personal Assistant" actor to the simple tasks "send new request" and "receive new results" (these use cases are *included* by the general use case). The included use cases of a use case that connects two agent roles are used to create liveness formulas in the agent interaction protocol *Process* field.

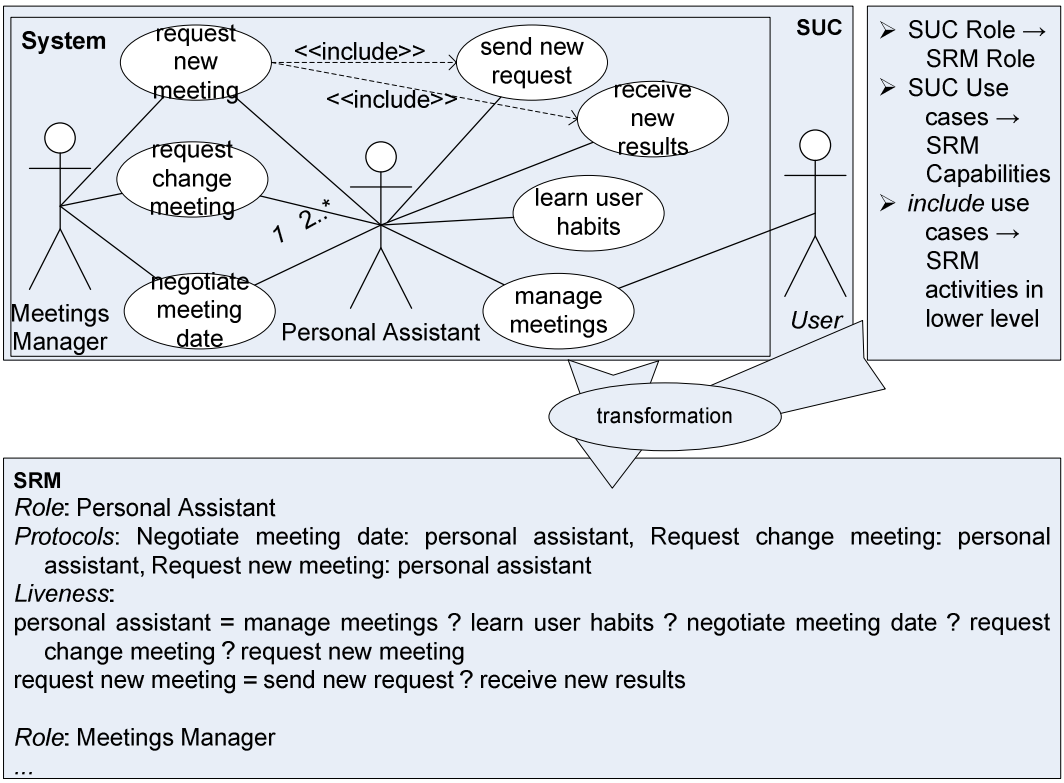


Figure 64: The SUC2SRM transformation.

Two AIP models for the meetings management system are presented in Table 4. The first is the "Request new meeting" discussed earlier and the second is a more complex model, the "Negotiate meeting date". Both define two roles, i.e. Personal Assistant and Meetings Manager, the rules for engaging (why would they participate

in this protocol), the outcomes that they should expect in successful completion and the process that they would follow in the form of a liveness formula. Any text editor can be used for defining the AIP model.

After refining the use cases, the SUC model is transformed to the system roles model (SRM). Figure 64 shows the transformation (SUC2SRM) process. A role model is created for each actor in the use case diagram. Each of the use cases connected to this role is transformed to an activity connected to others with a question mark. This activity can be automated if the analyst uses the Eclipse IDE with the SUC2SRM project presented in §5.2.1.2.

Table 4. Agent interaction protocols for the meetings management system

	Request new meeting		**Negotiate meeting date**	
Participants	*Personal Assistant*	*Meetings Manager*	*Personal Assistant*	*Meetings Manager*
Rules for engaging	He needs to create a meeting	-	He needs to create or participate to a meeting	He has a meeting with more than one participants that has no date assigned to it
Outcomes	He has received a confirmation of the meeting creation	-	He has scheduled participation to a meeting	He has arranged a meeting that met all the participants needs
Process	request new meeting = send new request. receive new results	request new meeting = receive new request. send new results	negotiate meeting date = receive proposed date. (decide response. send results. receive outcome)+	negotiate meeting date = (decide on date. send proposed date. receive results)+. send fixed date.

After the SUC2SRM transformation (shown in Figure 64) the formulas connect the activities with question marks. During the next ASEME activity, the analyst arranges the different capabilities in the liveness formulas in the right sequence and connects them with the appropriate Gaia operators (see Table 1), so that the formulas depict the process model of each role (see the SRM model in Figure 65). In the liveness property of the role, its name appears in the left hand side of the first formula (root formula). The use cases included by each capability are inserted as activities in a lower level formula (that has the capability on the left hand side). Thus, while in the SUC model the behavior (use cases) related to the roles is identified, in the SRM model the dynamic aspect of this behavior, i.e. what happens and when it happens, is defined. The *Sample Reflective Ecore Model Editor* of Eclipse or any XML editor can be used for graphically or textually (respectively) editing the SRM model.

The last activity of this ASEME phase defines the functionality table where the analyst associates each activity participating in the liveness formulas of the SRM to the technology or tool (functionality) that it will use (see Figure 66). Depending on the development iteration the functionalities can be vague and abstract (like the "machine learning" functionality) or concrete and final (like the "JADE FIPA AMS" and "argumentation-based decision making" functionalities). Depending on the choices made in this activity the risk related to the software development iteration may differ. Any text editor can be used for defining the Functionality Table (FT).

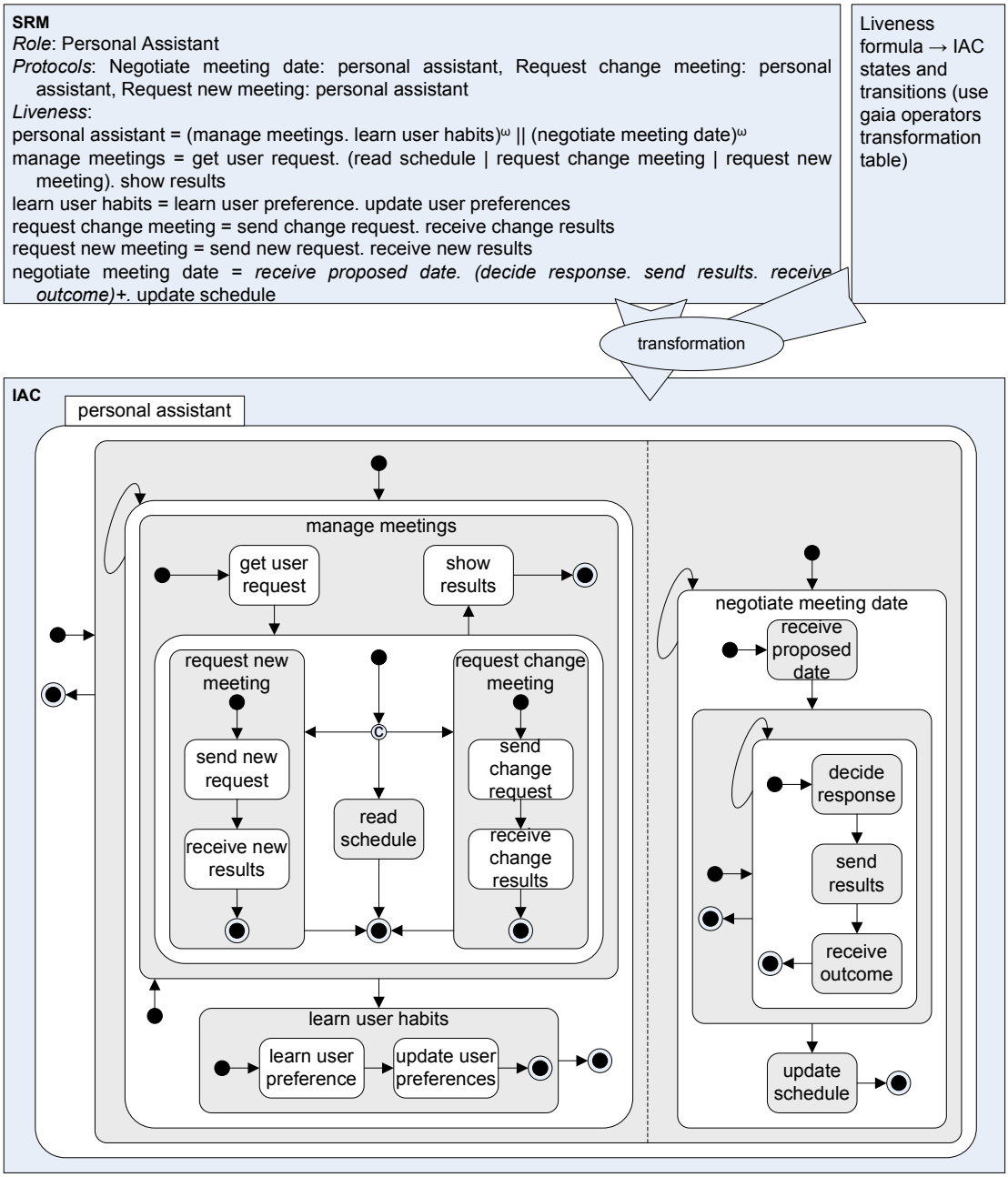


Figure 65. The SRM2IAC transformation.

Moreover, the identified technologies indicate the competences needed by the software development team. Finally, functionalities may be connected with different properties, such as programming language, execution environment and resources needed for their completion. For example the "argumentation based decision making" functionality can have the following properties:

- *Programming language*: Prolog

- *Execution environment*: SWI-Prolog[3] installed with Java interface (JPL)
- *Resources needed*: The Gorgias framework[4] and a knowledge base file

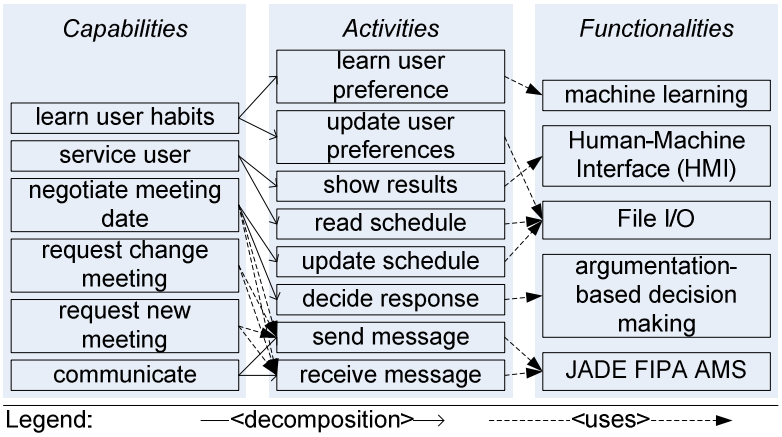


Figure 66. The Functionality Table for the personal assistant role of the meetings management system.

4.5 Design Phase

The ASEME design phase is presented in Figure 67. The three work definitions reflect the three different levels of abstraction in the software development. In the society level we have the inter-agent control model, in the agent level the intra-agent control model and in the capability level the models of the different components that will be used by the agent. Thus, each agent is considered to be part of a multi-agent system.

The agents communicate using interaction protocols that are described by the *inter-agent control* (EAC), which defines the participating roles and their responsibilities in the form of tasks. The agents implement the roles that they can assume through their capabilities. The capabilities are the modules that are integrated using the *intra-agent control* (IAC) concept.

The first work definition ("define inter-agent control model") of the design phase is detailed in Figure 68. It consists of four activities and produces four models.

The first activity, AIP2EAC, uses the "Gaia operators transformation templates" for transforming the process part of the agent interaction protocol model to a

[3] SWI-Prolog offers a comprehensive Free Software Prolog environment. Find out more in http://www.swi-prolog.org/

[4] Gorgias is a general argumentation framework that combines the ideas of preference reasoning and abduction. Find out more in http://www.cs.ucy.ac.cy/~nkd/gorgias/

statechart, namely the inter-agent control model (EAC). A state diagram is generated by an initial AND-state named after the protocol. Then, all participating roles define OR sub-states. The right hand side of the liveness formula of each role is transformed to several states within each OR-state by interpreting the Gaia operators in the way described in Table 5. This table has three columns. The first depicts a Gaia formula with a certain operator. The second shows how to draw the statechart relevant to this operator using the common statechart graphic language. The third shows how the same Gaia formula is transformed to the statechart representation defined in this thesis (as a tree branch).

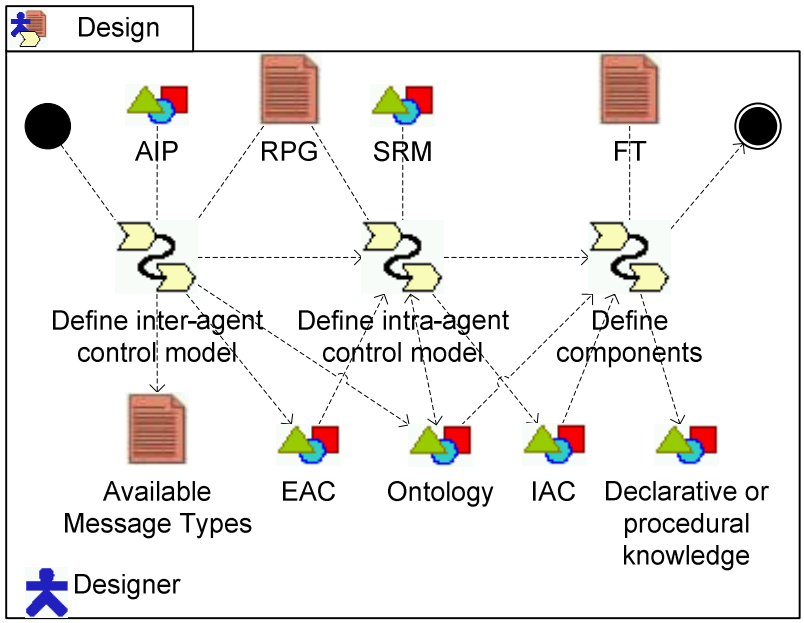


Figure 67. The ASEME Design Phase

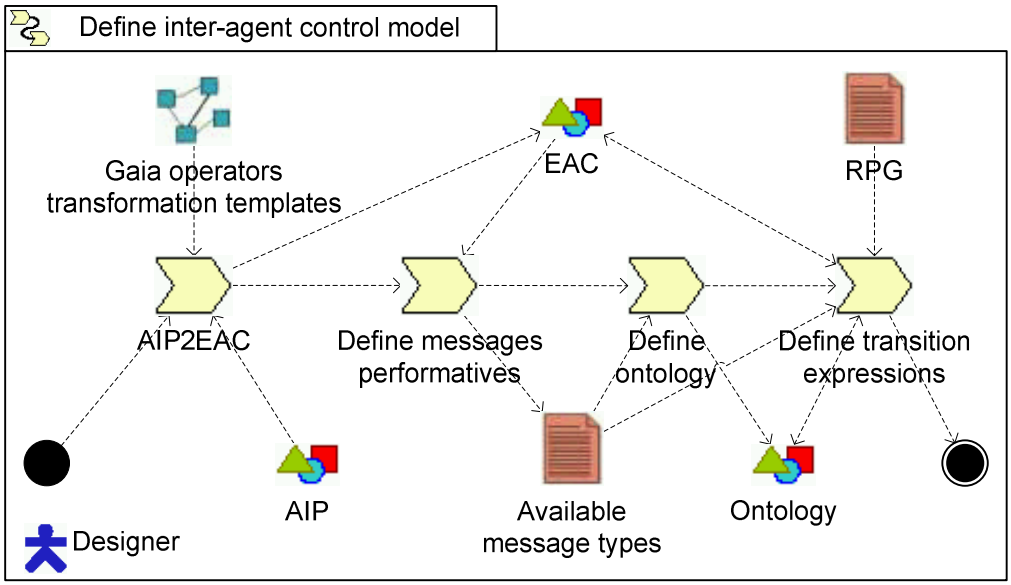
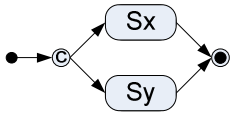
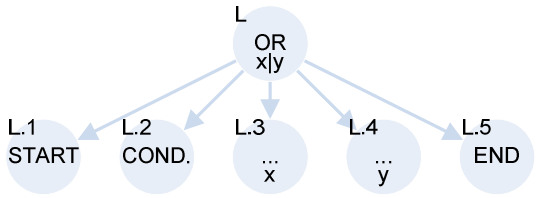
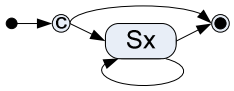
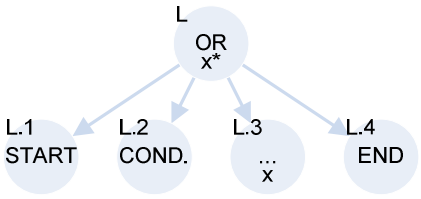
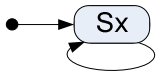
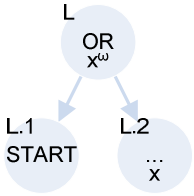
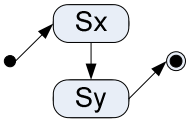
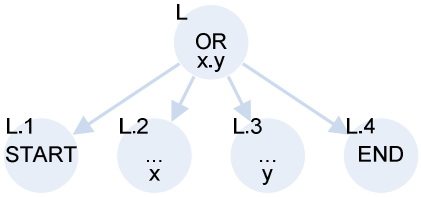
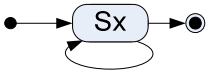
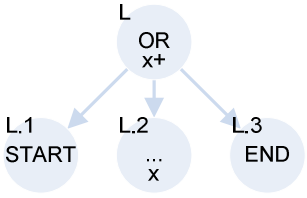
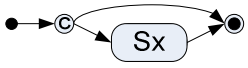
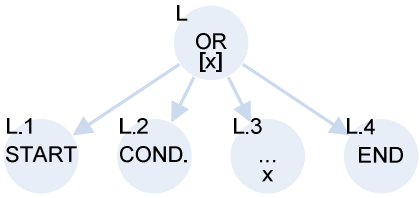
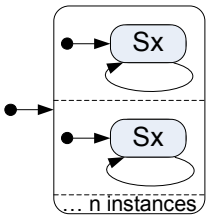
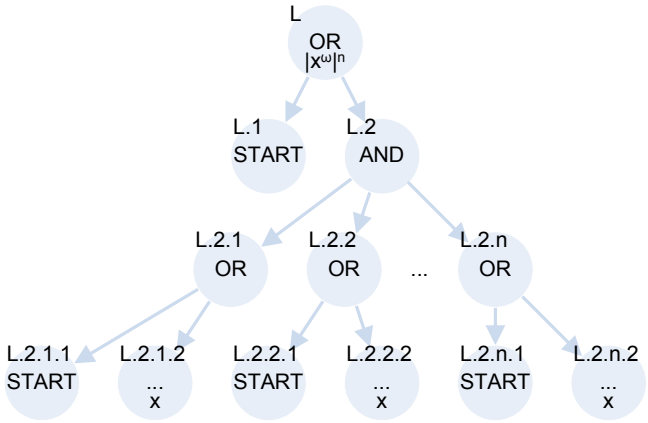
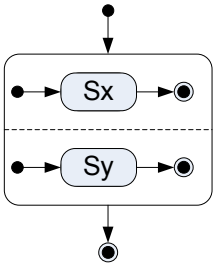
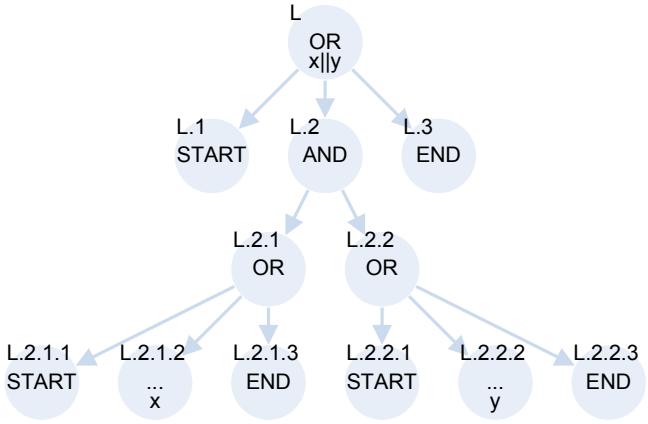


Figure 68: The "Define Inter-agent Control Model" work definition

The tree branch representation (in Table 5) uses grey arrows to connect a father node to its sons. On the top left of each node the label of the node is shown. The root node of each branch is supposed to have a label *L*and the other nodes are labeled accordingly. The type of each node is written centered in the middle of the node. Finally, the name of each node is centered in the bottom of each node. The reader should note that the nodes for the *x* or *y* variables of the Gaia formula do not have a node type. This is because it is possible that they are basic or non-basic nodes. If they are basic then the node's type is set to BASIC, otherwise another branch is added with this node as the root and as the reader can notice all templates set the type of the root of the branch.

Table 5. Templates of extended Gaia operators (Op.) for Statechart generation

Op.	*Template*	*Tree Branch*
x | y		
x*		
x^{ω}		
x . y		
x+		

Op.	*Template*	*Tree Branch*
[x]		
$\|x^{\omega}\|^{n}$		
x || y		

A designer can use the Gaia transformation templates to transform the liveness formula to a statechart. Alternatively, he can use an implementation of the recursive algorithm for building the statechart tree, which is presented in Listing 3 and it is executed in order to transform the liveness model to a statechart as it is defined in Definition 3.7 (see page 94). This algorithm is an important result of this thesis and the designer can use the Eclipse IDE with the SRM2IAC project presented in §5.2.2 for automating the transformation.

Listing 3. The transformation process from a liveness formula to a statechart in pseudocode.

```
Program transform(liveness)
  Var root = 0
  S = S ∪ {root}
  Name(root) = liveness->formula1->leftHandSide
```

```
  createStatechart(formula1->expression, root)
End Program

Procedure createStatechart(expression, father)
  var terms = 0
  for each termi in expression
    terms = terms + 1
  end for
  if terms > 1 then
    if expression is sequentialExpression then
      λ(father) = OR
      S = S ∪ {father.1}
      λ(father.1) = START
      var k=2
      for each termi in expression
        S= S ∪ {father.k}
        Name(father.k) = termi
        δ = δ ∪ {(father.(k-1), {}, father.k)}
        k = k + 1
      end for
      S = S ∪ {father.k}
      δ = δ ∪ {(father.(k-1), {}, father.k)}
      λ(father.k) = END
    else if expression is orExpression
      λ(father) = OR
      S = S ∪ {father.1}
      λ(father.1) = START
      S = S ∪ {father.2}
      λ(father.2) = CONDITION
      δ = δ ∪ {(father. 1, {}, father.2)}
      k = 3
      for each termi in expression
        S= S ∪ {father.k}
        Name(father.k) = termi
        δ = δ ∪ {(father.2, {}, father.k)}
        k = k + 1
      end for
      S = S ∪ {father.k}
      λ(father.k) = END
      var endNode = k
      k = k - 1
      while (k>2)
        δ = δ ∪ {(father.k, {}, father.endNode)}
        k = k – 1
      end while
    else if expression is parallelExpression
      λ(father) = OR
      S = S ∪ {father.1}
      λ(father.1) = START
      S = S ∪ {father.2}
      λ(father.2) = AND
      Name(father.2) = expression
      δ = δ ∪ {(father.1, {}, father.2)}
      S = S ∪ {father.3}
      λ(father.3) = END
      δ = δ ∪ {(father.2, {}, father.3)}
      k=1
      for each termi in expression
```

```
        S = S ∪ {father.2.k}
        λ(father.2.k) = OR
        Name(father.2.k) = "||"+term_i
        S = S ∪ {father.2.k.1}
        λ(father.2.k.1) = START
        S = S ∪ {father.2.k.2}
        Name(father.2.k.2) = term_i
        δ = δ ∪ {(father.2.k.1, {}, father.2.k.2)}
        S = S ∪ {father.2.k.3}
        λ(father.2.k.3) = END
        δ = δ ∪ {(father.2.k.2, {}, father.2.k.3)}
        k = k + 1
      end for
    end if
  for each term_i in expression
    if term_i is basicTerm
      handleBasicTerm(term_i, getNode(father, term_i)
    else
      if term_i is of type '('term')' then
        createStatechart(term, getNode(father, term_i))
      else if (term_i is of type '['term']') or (term_i is of type term'*') then
        λ(parent(getNode(father, term_i))) = OR
        S = S ∪ getNode(father, term_i).1
        λ(getNode(father, term_i).1) = START
        S = S ∪ getNode(father, term_i).2
        λ(getNode(father, term_i).2) = CONDITION
        S = S ∪ getNode(father, term_i).3
        Name(getNode(father, term_i).3) = term
        if term is basicTerm
          handleBasicTerm(term, getNode(father, term_i).3)
        else
          createStatechart(term, getNode(father, term_i).3)
        end if
        S = S ∪ getNode(father, term_i).4
        λ(getNode(father, term_i).4) = END
        δ = δ ∪ (getNode(father, term_i).1, {}, getNode(father, term_i).2)
        δ = δ ∪ (getNode(father, term_i).2, {}, getNode(father, term_i).3)
        δ = δ ∪ (getNode(father, term_i).2, {}, getNode(father, term_i).4)
        if term_i is of type term'*' then
          δ = δ ∪ (getNode(father, term_i).3, {}, getNode(father, term_i).3)
        end if
        δ = δ ∪ ( getNode(father, term_i).3, {}, getNode(father, term_i).4)
      else if (term_i is of type term'^ω') or (term_i is of type term'+') then
        λ(getNode(father, term_i)) = OR
        S = S ∪ getNode(father, term_i).1
        λ(getNode(father, term_i).1) = START
        S = S ∪ getNode(father, term_i).2
        Name(getNode(father, term_i).2) = term
        if term is basicTerm
          handleBasicTerm(term, getNode(father, term_i).2)
        else
          createStatechart(term, getNode(father, term_i).2)
        end if
        δ = δ ∪ (getNode(father, term_i).1, {}, getNode(father, term_i).2)
        δ = δ ∪ ( getNode(father, term_i).2, {}, getNode(father, term_i).2)
        if term_i is of type term'+' then
          S = S ∪ getNode(father, term_i).3
```

```
            λ(getNode(father, term_i).3) = END
            δ = δ ∪ (getNode(father, term_i).2, {}, getNode(father, term_i).3)
          end if
        else if term_i is of type '|'term'^ω|^n' then
          λ(getNode(father, term_i)) = AND
          For j=1 to n
            S = S ∪ getNode(father, term_i).j
            λ(getNode(father, term_i).j) = OR
            S = S ∪ getNode(father, term_i).j.1
            λ(getNode(father, term_i).j.1) = START
            S = S ∪ getNode(father, term_i).j.2
            Name(getNode(father, term_i).j.2) = term
            if term is basicTerm
              handleBasicTerm(term, getNode(father, term_i).j.2)
            else
              createStatechart(term, getNode(father, term_i).j.2)
            end if
            δ = δ ∪ (getNode(father, term_i).j.1, {}, getNode(father, term_i).j.2)
            δ = δ ∪ (getNode(father, term_i).j.2, {}, getNode(father, term_i).j.2)
          End for
        end if
      end if
    end for
end function

Function getNode(father, term)
    QueuedList queue
    queue.addLast(father)
    Do while queue.notEmpty()
      element_i = queue.getFirst()
      If Name(element_i) = term then
        Return element_i
      Else
        For each son_j in sons(element_i)
          queue.addLast(son_j)
        end for
      End if
    end do
end function

Function handleBasicTerm(term, node)
    Var isBasic = true
    For each formula_i in liveness
      If (formula_i->leftHandSide = term)
        createStatechart(formula_i->expression, node)
        isBasic = false
      End if
    end for
    If isBasic
      λ(node) = BASIC
    end if
end function
```

The liveness model for the EAC model for a protocol named *protocol_name* including *n* roles is the:

protocol_name = (role 1 process) || (role 2 process) || … || (role n process)

For the case of the meetings management system the liveness formula for the negotiate meeting protocol is the (using the defined processes in Table 4):

negotiate_meeting_protocol = *(receive proposed date. (decide response. send results. receive outcome)+) || (decide on date. send proposed date. receive results)+. send fixed date)*

After applying the transformation algorithm, the statechart depicted in Figure 69 is created.

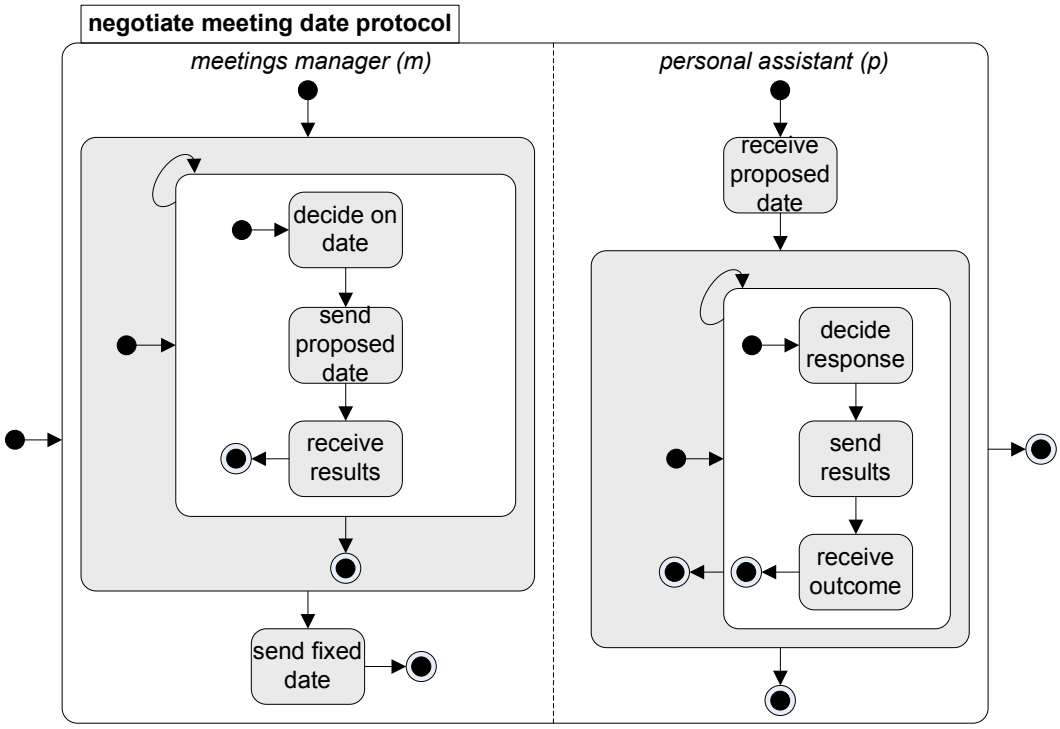


Figure 69. The automatically generated EAC model for the "Negotiate Meeting Date" protocol.

Then, in the next activity, the designer defines the message performatives allowed within the protocol. A message is expressed by *P(x,y,c)* where *P* is the *performative*, *x* is the sender, *y* the receiver and *c* the message body. For the meetings management system, $P \in \{accept, propose, reject, inform\}$. The designer can use any text editor for defining the message types.

The items that the designer must define at the next activity are the data structures used for defining the protocol parameters (also referred to as the ontology), the timers and the message contents (also part of the ontology). A first issue in the ontology is how to reference agents. According to the FIPA standard (FIPA TC Agent Management, 2004) an agent is identified by the *Agent Identifier* (AID) that contains its name, a list of addresses and a list of resolvers. For the case of software agents the use of FIPA standards can save a developer a lot of time. The ontology for the

meetings management system is presented in Figure 70. The designer can use a number of existing tools or frameworks for producing the ontology. UML class diagrams (as in Figure 70 for the meetings management system) or the Protégé[5] ontology editor (as will be shown later in the MARKET-MINER case study) are examples of graphical tools that can be used at this activity.

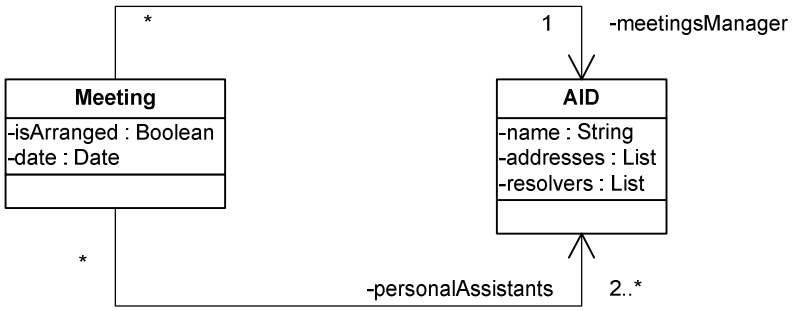


Figure 70. The Ontology for the meetings management system.

Finally, in the last activity of the "Define Inter-agent Control Model" work definition, the transition expressions are defined. The resulting statechart for the *negotiate meeting date protocol* (starting from the analysis model presented in Table 4) is depicted in Figure 71.

The preconditions of the agent interaction protocol become the conditions of a transition from a START state that targets the first state of the protocol for each role. The preconditions for the meetings manager role, which have been described in free text in the agent interaction protocol model (see *rules for engaging* in Table 4), concern the arrangement of a meeting with a list of more than one participants (personal assistants). They become the conditions for the transition that has as source the START state within the meetings manager OR state. The conditions are that the *Meeting* type (see the *Meeting* type definition in Figure 70) variable *meeting* has the property *isArranged* equal to *False* and its *personalAssistants* list has more than one entries. The first condition is expressed in logic-based format, i.e. isArranged(meeting) = False, using the procedure that will be defined in the case study later in this chapter (see §4.9.3) for transforming an ontology expressed in object-oriented format (like in Figure 70) to logic-based format.

It is worth taking a close look to Figure 71 as it depicts a more complex protocol than the one presented earlier in Figure 51 and one that implicates more than two participants. There is a meetings manager role and more than one personal assistant roles. Let's see how the meetings manager role can manage this situation. If his preconditions are met (as was shown in the previous paragraph), he enters the large

[5] Protégé is a free, open source ontology editor and knowledge-base framework. Find out more in http://protege.stanford.edu/

grey OR state. The START state causes a transition to the white OR state automatically as there is no transition expression and again, using an automatic transition from the START state the meetings manager enters the "decide on date" state.

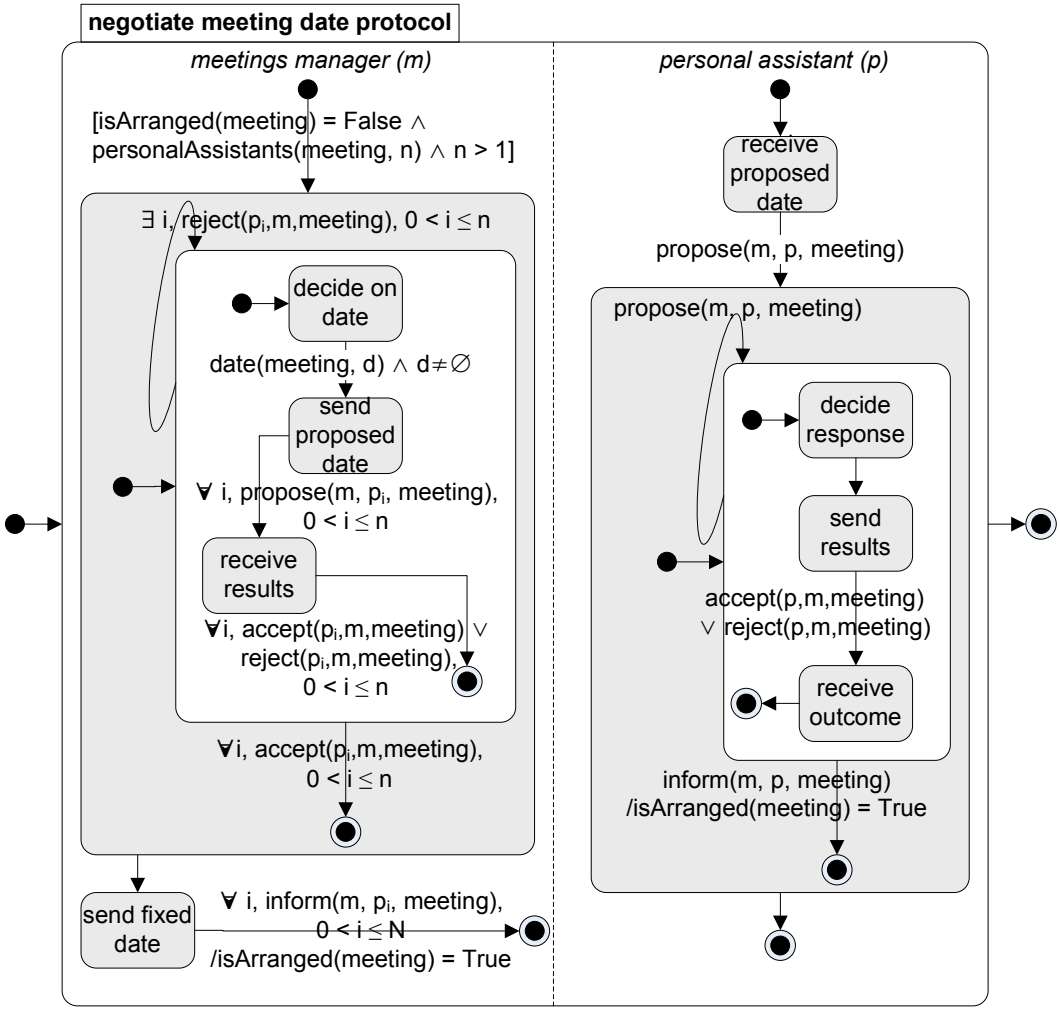


Figure 71. The complete EAC model for the "Negotiate Meeting Date" protocol.

The event date(meeting, d) ∧ d≠∅ implies that the transition to the "send proposed date" state will only be taken after the *date* property of the *meeting* variable has been assigned a value by the activity of the "decide on date" state. As soon as this is achieved, the "send proposed date" state is entered. The event ∀i, propose(m, p_i, meeting), 0 < i ≤ n shows that this state can only be exited after *n* messages with performative *propose* have been sent to the *n* different personal assistant participants of this protocol (and meeting). After sending the messages the meetings manager takes the transition to the "receive results" state. This state is itself exited if *n* messages with performative either *Reject* or *Accept* have been received.

After the last transition, the white OR state must be exited. The reader can see that there are two possibilities. One is towards the END state that is taken on the event that all the personal assistants have sent messages with the *Accept* performative. If, however there is even one personal assistant that has sent a message with the *Reject* performative this state is re-entered and the procedure is repeated until all the personal assistants reply with the *Accept* performative.

In the latter case the "send fixed date" state is entered. The activity of this state is about sending a last *Inform* message to all the personal assistant agents participants, which informs them about the final agreed meeting date. This is the enabling event of the transition exiting this state and, before finishing, the predicate *isArranged* is evaluated to *True*. The same action is undertaken by the final transition of the personal assistant's orthogonal component. Thus, in a successful completion of the protocol both the personal assistants and the meetings manager have an arranged agreed upon meeting.

The second work definition of the ASEME design phase, i.e. "Define intra-agent control model", is detailed in Figure 72. The intra-agent control is created by transforming the liveness model of the role (SRM) to a state diagram (IAC). This is achieved by interpreting the Gaia operators in the way described in Table 5. Initially, the statechart has only one state named after the left-hand side of the first liveness formula of the role model. Then, this state acquires substates. The latter are constructed by reading the right hand side of the liveness formula from left to right, and substituting the operator found there with the relevant template in Table 5. If one of the states is further refined in a next formula, then new substates are defined for it in a recursive way. The transformation process (SRM2IAC) is shown graphically in Figure 65. The designer can use the Eclipse IDE with the SRM2IAC project presented in §5.2.2 for automating the transformation.

The next activity imports the transition expressions from the inter-agent control (EAC) for the part of the statechart containing a protocol (i.e. the part of the statechart produced from the formula whose left hand side is a protocol capability). For example, the expression for the transition from the activity "send results" to the activity "receive outcome" in Figure 65 (IAC) must be the same as the one in Figure 71 (EAC). Thus, the designer must copy the transition expression from the EAC model to the IAC model. The *Sample Reflective Ecore Model Editor* of Eclipse or any XML editor can be used for graphically or textually (respectively) editing the IAC model.

The last activity of this work definition is about enriching the rest of the statechart with transition expressions, updating the ontology if necessary. Again, the *Sample Reflective Ecore Model Editor* of Eclipse or any XML editor can be used for editing the transition expressions of the IAC model.

The final work definition of the ASEME design phase (i.e. "Define components", the last work definition shown in Figure 67) is about designing the activities that are executed in each state. The input needed is the "Functionality table" to indicate the technology (e.g. which library to import and which programming language to use), the "Ontology" to show the data structures that will be used by this activity and the "Intra-agent control model" that lists all the activities as states. The output depends

on the technology used for each activity and can be declarative or procedural knowledge (or both).

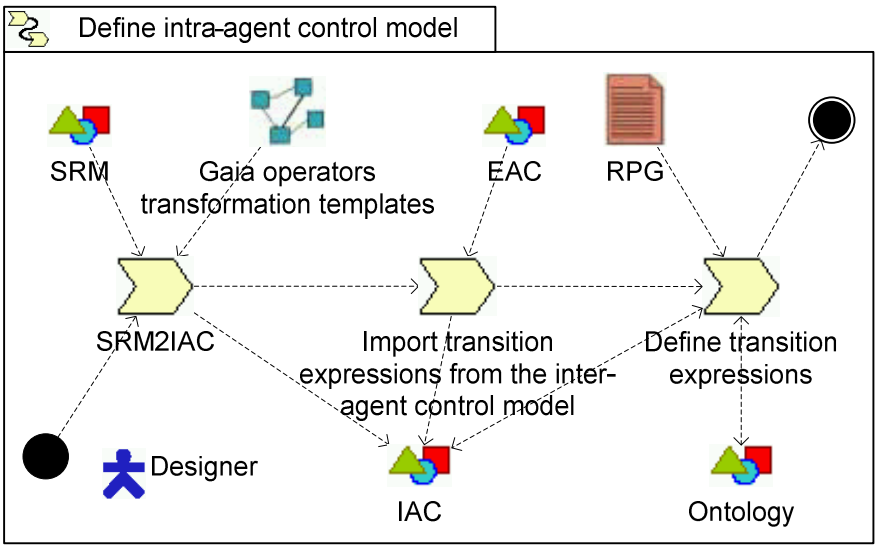


Figure 72: The "Define Intra-agent Control Model" work definition

4.6 Implementation Phase

The implementation phase's goal is to transform the platform independent model to a platform dependent model, i.e. an implementation model. This phase can have different instantiations according to the implementation platform. The implementation phase details a transformation process of the PIM to PSM and then to a computer program. The IAC model can be transformed to any language that is supported by a statecharts-based case tool. Rhapsody, for instance can transform the statechart model to C++ and Java code. However, it is important to provide a transformation process for an agent development platform as the ASEME process is about agent development.

In Chapter 5 (§5.2.3.2) the transformation process of the IAC model to agent code using the JADE agent development platform is presented in detail. Using this process the developer can automatically generate all JADE agent and behavior classes that will be needed along with the classes representing the IAC model used variables. Moreover, a large part of the needed code is automatically generated, or even the totality of the code depending on the behaviour type. The JADE platform was selected for demonstrating the capability to transform the IAC model to an agent implementation for several reasons:

a) Bordini et al. (2006), in their survey of programming languages and platforms for Multi-Agent Systems, found that the most popular agent platform is JADE

b) It is open source, thus available to anyone, a fact which increases the potential visibility of this work

c) It has an excellent users community (about 4.5 messages per day in the *jade-develop* mailing list for the year 2008) that helps new users to achieve their goals quickly (minimizing learning time) and that continuously provides new add-ons

d) It is compliant with the FIPA standards

e) It can be compiled for use on devices with limited resources such as PDAs and mobile phones

However, the last paragraph of this chapter (4.9) also shows the usage of the commercial Rhapsody tool for developing a Java language statecharts-based prototype modeled by ASEME.

4.7 Verification and Optimization Phases

During the Verification Phase the system's functionality is verified in comparison to its requirements. The verification phase can be carried out in parallel for the three different abstraction levels; however, the best approach is sequential: the software components are tested for the successful implementation of algorithms, the agents for the successful implementation of capabilities and the MAS for its overall correct operation.

The Optimization Phase is concerned with the optimization of the system. The algorithms in the capability level can be optimized in execution time or resource consumption. In the agent level, the number of concurrently executing capabilities can be optimized, i.e. how many instances of the "negotiate meeting date" capability should the personal assistant be executing concurrently to meet his requirements. Finally, in the societal level, the number of agents that will be instantiated and the strategy for instantiating or destroying agents while the system is in operation is optimized.

Both verification and optimization can be related to the concept of scalability. Scalability (see Rana and Stout, 2000) is defined as the ability of a solution to work when the size of the problem increases. A MAS can scale when the number of agents in a platform increases or when the number of agents among several platforms increases. To our knowledge the current AOSE methodologies rarely address the issue and do not provide documented solutions. Sturm and Shehory (2004) after performing an evaluation of AOSE methodologies concluded that scalability is not supported by the methodologies.

In Chapter 6, a process model based approach for verification and optimization is presented in detail. Moreover, the usage of this approach for the addressing of a scalability issue in a real world system development process is presented as a case study.

4.8 Support for Sub-dialogs

The work of Moore (2000) supports the possibility of an agent getting involved in a sub-dialog when in a dialog. In AMOLA, the model for describing such dialogs is the inter-agent control model (EAC). Moore supposed that the agent has access to a repository of dialogs and dynamically selects a sub-dialog model whenever an incoming message is not permitted by the existing dialog but is permitted by another in the repository. In this section we show how AMOLA models such sub-dialogs.

AMOLA models use the following assumption: The role that starts a protocol (i.e. sends the first message) assigns a conversation identification string (CIS) to it. This token is also used by all following messages in the same protocol (or dialog). When a message arrives, the agent checks its CIS and allocates it to the relevant *receive message* activity for processing. This activity, however, will only use this message if it triggers a state transition. If the message is not of the type expected, then it is not used. By defining the way that such a message will be handled, sub-dialogs can be allowed within dialogs: A message with a specific CIS that is not used by the relevant protocol activity is treated as a new protocol message and is, thus, forwarded to all activities that wait for similar performative messages. The first activity that uses it gets priority for the specific CIS until its protocol finishes.

Let's provide an example. Imagine that the meetings manager loses the scheduled meetings data and needs to ask (after a first failed exchange of messages) the participants of a new meeting for their schedules. This question is not foreseen by the "Negotiate Meeting Date" protocol (see Figure 71). The personal assistant presented in Figure 65 will disregard this question (he is just waiting for *propose* messages). However, the *personal assistant* (PA) depicted in Figure 73 will be able to answer the request. This PA implements another protocol, the "Provide Schedule Information" protocol (on the right side of the figure) that is a simple protocol that waits for a message, reads the user's schedule and sends it back to the requester.

Being in the receive outcome sub-state of the negotiate meeting date protocol state the PA waits either an *inform* or *propose* message from the Meetings Manager (consult Figure 71 for the transition expressions). The *request* message for the user's schedule will arrive with the same CIS. The *receive outcome* state gets first the chance to process this message, however it disregards it (is not a legal response). Then the next activity that waits for a message, *receive schedule request* gets a chance to process it. The activity finds that it is a legal message asking for the user's schedule. It processes it and then finishes. The *Meetings Manager* sends the *inform*

message as he found a date for the meeting and the *receive outcome* state finally catches the response it has been waiting for.

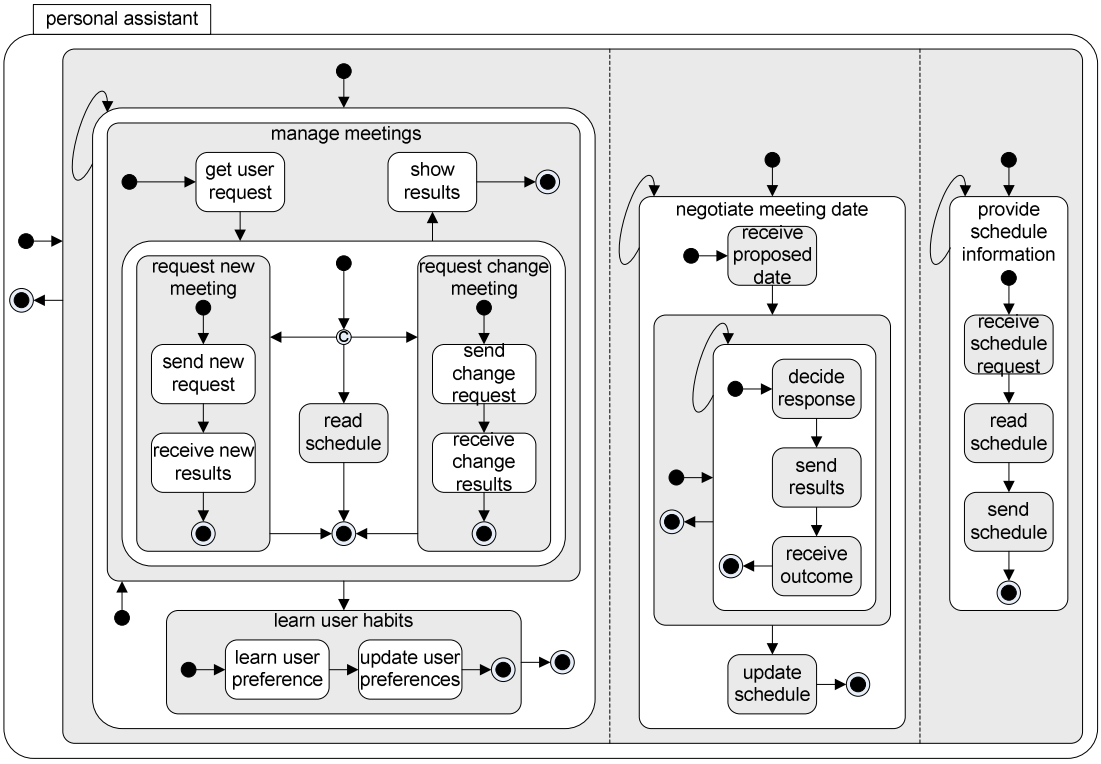


Figure 73. The intra-agent control model supporting a sub-dialog

Thus, AMOLA caters for implementation of the theoretical concept of sub-dialogs as it has been proposed by Moore. This is another originality of this thesis. The JADE framework allocates messages using the policy described here. Therefore, if the developer chooses to use JADE for the AMOLA models implementation (as it is described in Chapter 5) then he can use sub-dialogs.

4.9 A Case Study: The MARKET-MINER project

The MARKET-MINER project is of great interest as a case study for ASEME and AMOLA, as it demonstrates the capability of AMOLA to create models that will be developed using just an object-oriented programming language such as Java and not a specialized agent-oriented platform. Moreover, it demonstrates the development of an agent that uses argumentation for decision making, a logic-based technology. Firstly, the MARKET-MINER agent's requirements are presented so that the reader understands what the project aimed in achieving. A brief description of the used argumentation framework follows in order to show how the agent would perform its

decision making. Then, the domain knowledge modeling procedure is presented. This procedure is interesting, in a knowledge engineering point of view, because it was developed using a popular open source tool and was used in object-oriented format (for use by the different agent modules) but also in logic format (for use by the argumentation framework). The process of transforming the object oriented ontology to a logic-based one is presented in detail. Finally, the ASEME process steps followed are presented in detail demonstrating its use.

4.9.1 The MARKET-MINER Project. An Introduction

Automating the product pricing procedure in many different types of enterprises like retail businesses, factories, even firms offering services is an important issue. Product pricing is concerned with deciding on which price each of a firm's products will have in the market. The product pricing agent that is presented in this case study allows for the integration of the views of different types of decision makers (like financial, production, marketing officers) and can reach a decision even when these views are conflicting. This is achieved with the use of argumentation.

Argumentation has been used successfully in the last years as a reasoning mechanism for autonomous agents in different situations, as for example for deliberating over the needs of a user with a combination of impairments (Moraitis and Spanoudakis, 2007) and for selecting the funds that should be included in an investment portfolio (Spanoudakis and Pendaraki, 2007c). It is the first time that it was used for decision making in the retail business sector. Argumentation responded well to the MARKET-MINER requirements, which demanded a system that would have the possibility to apply a pricing policy adjusted to the market context, in the meanwhile reflecting the points of views of diverse decision makers.

4.9.2 The Argumentation Framework

Decision makers, be they artificial or human, need to make decisions under complex preference policies that take into account different factors. In general, these policies have a dynamic nature and are influenced by the particular state of the environment in which the agent finds himself. The agent's decision process needs to be able to synthesize together different aspects of his preference policy and to adapt to new input from the current environment. The product pricing decision maker was modeled as such an agent.

In order to address requirements like the above, Kakas and Moraitis (2003) proposed an argumentation based framework to support an agent's self deliberation process for drawing conclusions under a given policy. The Gorgias open source framework based on the Prolog language provides an implementation for the framework of Kakas and Moraitis. It was the framework that was chosen for modeling the decision making functionality of the MARKET-MINER product pricing agent.

Gorgias defines a specific language for the object level rules and the priorities rules of the second and third levels. A negative literal is a term of the form *neg(L)*. The language for representing the theories is given by rules with the syntax:

```
rule(Signature, Head, Body)
```

In this rule syntax, *Head* is a literal, *Body* is a list of literals and *Signature* is a compound term composed of the rule name with selected variables from the Head and Body of the rule. The predicate *prefer/2* is used to capture the higher priority relation *(h_p)* defined in the theoretical framework. It should only be used as the head of a rule. Using the previously defined syntax we can write the rule:

```
rule(Signature, prefer(Sig1, Sig2), Body).
```

This rule means that the rule with signature *Sig1* has higher priority than the rule with signature *Sig2*, provided that the preconditions in the *Body* hold. If the modeler needs to express that two predicates are conflicting he can express that by using the rule:

```
conflict(Sig1,Sig2).
```

This rule indicates that the rules with signatures *Sig1* and *Sig2* are conflicting. A literal's negation is considered by default as conflicting with the literal itself.

4.9.3 Domain Knowledge Modeling

Firstly, the domain knowledge was gathered in free text format by questioning the decision makers that participate in the product pricing procedure. They were officers in Financial, Marketing and Production departments of firms in the retail business but also in the manufacture domain. Then, their statements were processed aiming on one hand to discover the domain ontology and on the other hand the decision making rules. For example, let's consider the expression "If the firm has a high-low strategy then if it advertises a product and its price is low the products that accompany it in the consumers' basket are priced high". This expression identifies the concepts "firm strategy" and "product". The concept firm strategy can have the property "high-low" and the product can have the property "price" and can be related to other products as "accompanied in the consumer's basket by" them.

The next step was to ask a team of decision makers to decide on priorities between the different conflicting extracted rules. These priorities could be default or dependent on context.

The Protégé ontology editor was used for defining the domain concepts and their properties and relations. In Figure 74, the *Product* concept and its properties are presented. The reader can see the properties identified previously *hasPrice* and *isAccompaniedBy*. Price is defined as a real number (*Float*) and *isAccompaniedBy* relates the product to multiple other instances of products that accompany it in the consumer's cart. In the figure, the firm strategy concept is also presented. Its properties are all *Boolean* and represent the different strategies that the firm can

have activated at a given time. For example, the *hitCompetition* property is set to *true* if the firm's strategy is to reduce the sales of its competitors. The property *retail_business* characterizes the firm as one in the retail business sector.

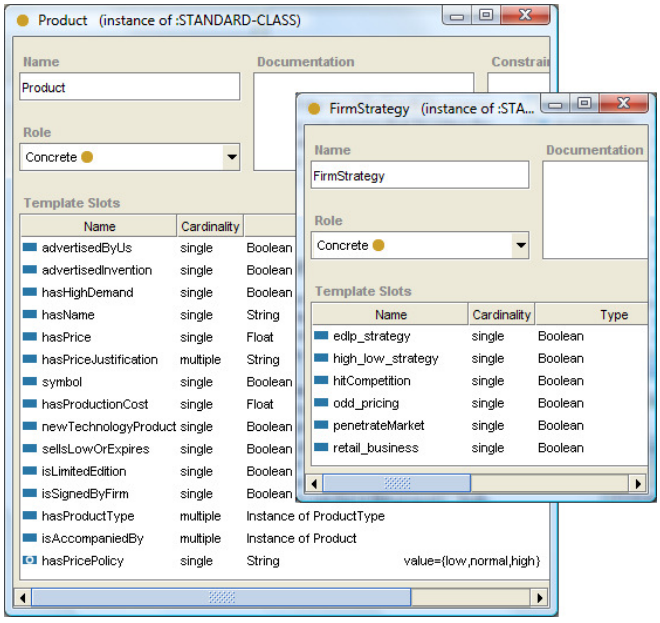


Figure 74. The MARKET-MINER *Product* and *FirmStrategy* ontology concepts.

The knowledge base definition, however, needed to be expressed in logic-based format as the Gorgias argumentation framework uses the Prolog language. The following procedure must be used in order to use the concepts and their properties as they are defined in Protégé:

- A Boolean property is encoded as a unary predicate, for example the *advertisedByUs* property of the *Product* concept is encoded as *advertisedByUs(ProductInstance)*.
- A property with a string, numerical, or any concept instance value is encoded as a binary predicate, for example the *hasPrice* property of the *Product* concept is encoded as *hasPrice(ProductInstance, FloatValue)*.
- A property with a string, numerical, or any concept instance value with multiple cardinality is encoded as a binary predicate in two ways:
    - The first possibility is for the second term of the predicate to be a list. Thus, the *isAccompaniedBy* property of the *Product* concept is encoded as *isAccompaniedBy(ProductInstance, [ProductInstance1, ProductInstance2, …])*, where product instances must not refer to the same product.
    - A second possibility is to create multiple predicates for the property. For example the *hasProductType* property of the Product concept is

encoded as *hasProductType(ProductInstance, ProductTypeInstance)*. In the case that a product has more than one product types, one such predicate is created for each product type.

Then, the Gorgias framework was used for writing the rules. The goal of the knowledge base would be to decide on whether a product should be priced high, low or normally. Thus it emerged, the *hasPricePolicy* property of the *Product* concept. After this decision, the object-level rules were written, each having as head the predicate *hasPricePolicy(Product, Value)* where *Value* could be *low*, *high* or *normal* – the relevant limitation for this predicate is also defined in the ontology (see the *hasPricePolicy* property of the *Product* concept in Figure 74).

Then, the different policies were defined as conflicting, thus only one policy was acceptable per product. In order to resolve conflicts, the firm (executive) officers were consulted. They defined priorities over the conflicting object rules. Consider, for example, the rules presented in Listing 4.

Listing 4. An extract of the Gorgias rules for the MARKET-MINER project. Variables start with a capital letter as in Prolog.

```
…
rule(r1_2_2(Product), hasPricePolicy(Product, low),
  [hitProductTypeCompetition(ProductType), hasProductType(Product, ProductType)]).

rule(r2_3(Product), hasPricePolicy(Product, high),
  [newTechnologyProduct(Product), advertisedInvention(Product)]).

rule(pr1_2_6(Product), prefer(r1_2_2(Product), r2_3(Product)), []).
…
```

Rules *r1_2_2* and *r2_3* are conflicting if they are both activated for the same product. The first states that a product should be priced low if the firm wants to hit the competition for its product type, while the second states that a new technology product that is an advertised invention should be priced high. To resolve the conflict the *pr1_2_6* priority rule is added, which states that *r1_2_2* is preferred to *r2_3*.

4.9.4 The Product Pricing Agent

This section presents the market-miner product pricing agent (also referred to as MIPA) development process. Then two important aspects of it are presented in detail, the decision making module and the human-computer interaction.

MIPA (the market-miner product pricing agent) was engineered using ASEME. During the analysis phase, the actors and the use cases related to the agent-based system were identified (see Figure 75). The system actor is MIPA, while the external actors that participate in the system's environment are the user, external systems of competitors, weather report systems (as the weather forecast influences product

demand as in the case of umbrellas) and municipality systems (as local events like concerts, sports, etc, also influence consumer demand). The actors were too few to need to start with a requirements analysis phase. Firstly, the general use cases (like *interact with user*) were identified and they were then elaborated in more specific ones (like *present information to the user* and *update firm policy*) using the <<include>> relation.

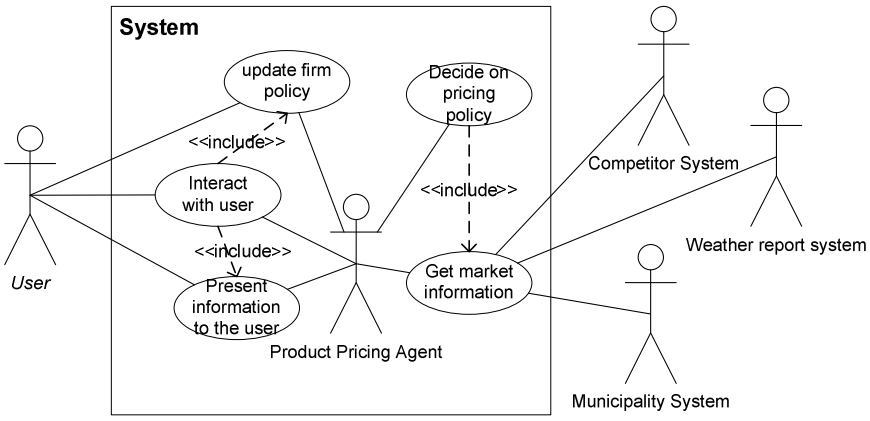


Figure 75. MIPA Use Case Diagram

Then, the roles model was created, as it is presented in Figure 76(a). This model defines the dynamic aspect of the system, general use cases were transformed to capabilities, while the specific ones were transformed to activities.

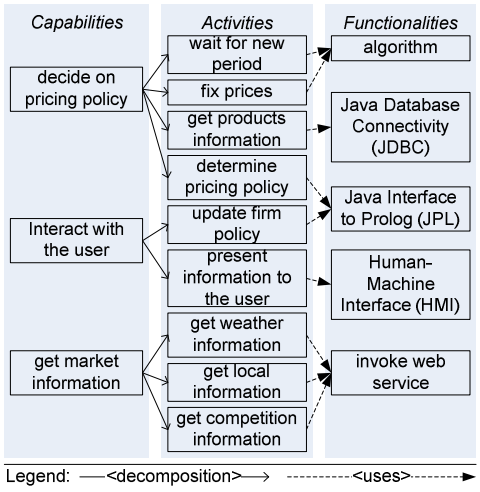
The next step was to associate each activity to a functionality, i.e. the technology that would be used for its implementation. In Figure 76(b) the reader can observe the capabilities, the activities that they decompose to and the functionality associated with each activity. The choice of these technologies is greatly influenced by non-functional requirements. For example the system will need to connect on diverse firm databases. Thus, the JDBC[6] technology was selected, as it is database provider independent.

The last step, before implementation, is to extract from the roles model the statechart that resembles the agent. This is achieved by transforming the liveness formula to a statechart in a straightforward process that uses templates to transform activities and Gaia operators to states and transitions (see the transformation process in §4.5 for more details). The resulting statechart, i.e. the intra-agent control (as it is called in ASEME) is depicted in Figure 77. It was defined in the Rhapsody CASE tool. This tool automates the process of transforming a statechart to C++, Java, C and Ada code.

[6] The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases – SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access. Find out more in http://java.sun.com/javase/technologies/database/

Role: Product Pricing Agent
Liveness:
product pricing agent = (decide on pricing policy)$^{\omega}$ || (interact with user)$^{\omega}$ || [(get market information)$^{\omega}$]
decide on pricing policy = wait for new period. get products information. determine pricing policy. fix prices.
interact with user = (present information to the user | update firm policy)+
get market information = get weather information. get local information. get competition information. update facts

(a)



(b)

Figure 76. MIPA Role Model (a) and the relation between Capabilities, Activities and Functionalities (b).

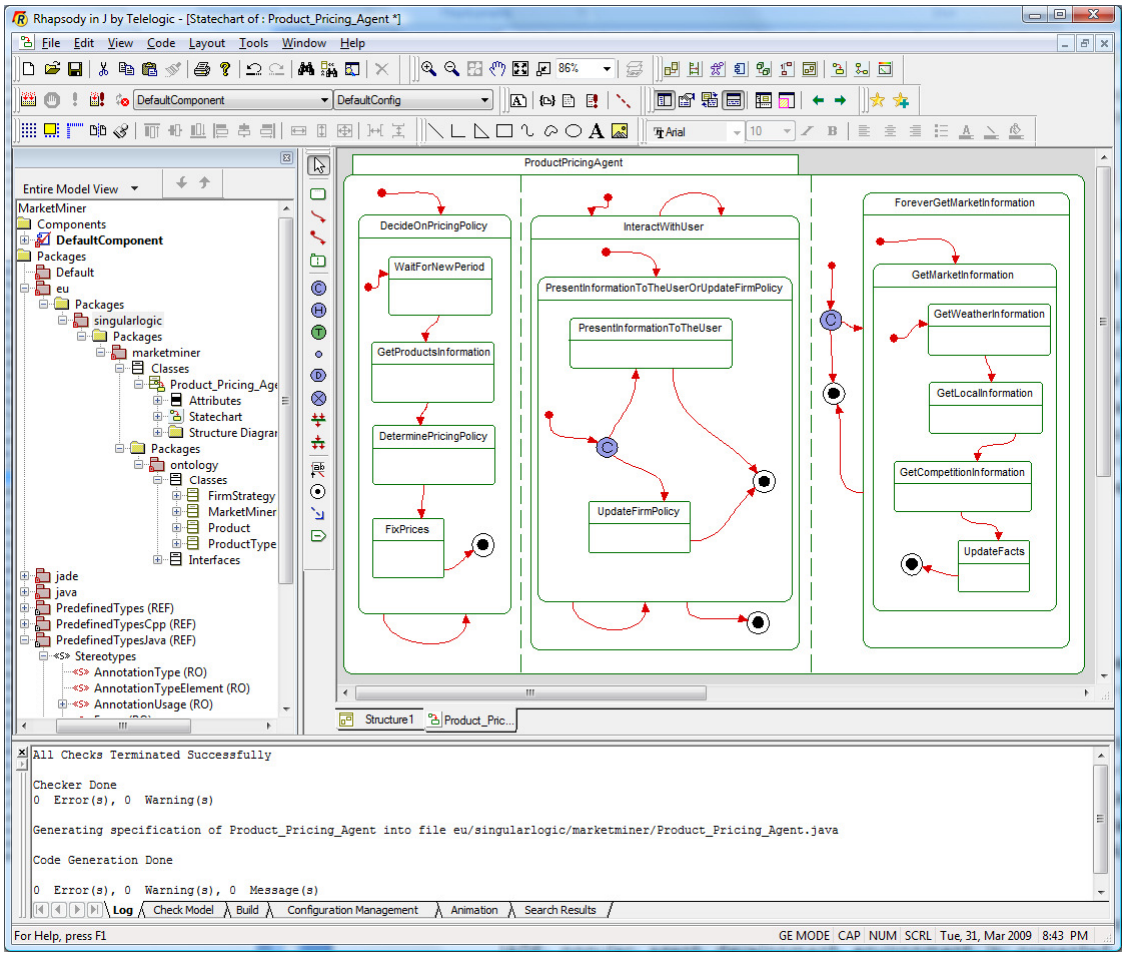


Figure 77. MIPA Intra-agent Control Model (snapshot from the Rhapsody© CASE tool).

In Listing 5 an extract from the automatically generated code for MIPA is presented. It is the place where the programmer is instructed to write the code for connecting to the web service for getting products information as the entry activity of the state. In the relevant java method's comments the reader can also observe the position of the specific state activity in the states hierarchy as it is represented by the Rhapsody© tool (*ROOT. ProductPricingAgent. ForeverDecideOnPricingPolicy. DecideOnPricingPolicy. GetProductsInformation*). The automatically generated code for the MIPA intra-agent control is included in its totality in Annex 7.

Listing 5. An extract from the automatically generated Product_Pricing_Agent java class by the Rhapsody© CASE tool.

```
…
      public void GetProductsInformationEnter() {
         //#[ state
ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingPolicy.GetProducts
ctsInformation.(Entry)
         //connect to web service
         //#]
      }
…
```

The details of the implementation beyond this point are not relevant to this thesis; however, the reader can find more information about the implementation in Spanoudakis and Moraitis (2008c and 2009).

4.9.5 Evaluation

The product pricing agent application was evaluated by SingularLogic SA (http://www.singularlogic.eu), the largest Greek software vendor for SMEs. The Software business unit is involved in the development and provision of business software products for the SME market, the provision of services (implementation and adaptation of applications, training and maintenance services), as well as the promotion and support of products by third parties, both in the entirety of the Greek market and the Balkan markets. The unit's software applications are trusted by 40,000 businesses both in Greece and abroad.

The MARKET-MINER project included the application analysis, design, implementation and evaluation phases. It also produced an exploitation plan (Toulis et al., 2007a). The application evaluation goals were to measure the overall satisfaction of its users. In the evaluation report (Toulis et al., 2007b) three user categories were identified, System Administrators, Consultants and Data Analysts.

At this point the reader should note that the MARKET-MINER project had a wider scope than that of the product pricing agent, therefore this paragraph will focus in

the part of the study relevant to it - the pricing application. Thus, only the Consultants and System Administrators user categories are relevant (data analysts were engaged in the data mining module of MARKET-MINER that is beyond the scope of this thesis).

The following criteria were used for measuring user satisfaction:

- *Performance* (C1): This criterion measures the capability of the system to produce valid and accurate results.
- *Usability* (C2): This criterion measures the satisfaction of the user with regard to his experience in using the system, including the training phase and the ease of achieving his tasks.
- *Interoperability* (C3): MARKET-MINER depends heavily on its seamless integration with legacy systems databases. Thus we needed to measure the openness of the system or the efficiency of connecting it to the existing databases.
- *Security and Trust* (C4): MARKET-MINER accesses enterprise databases and handles sensitive information relevant to the firm's market strategy. Thus, it is important that the user feels that the data are securely handled and remain confidential.

The users expressed their views in a relevant questionnaire where each criterion was presented with several sub-criteria and they marked their experience on a scale of one (dissatisfied) to five (completely satisfied) and their evaluation of the importance of the criterion on a scale of one (irrelevant) to five (very important). The evaluation was based on 25 questionnaires, 15 of which were completed by decision makers (with financial background), seven by data analysts (computer science background) and three by system administrators.

The consultants were experienced in applying business intelligence solutions to enterprises mostly in the retail sector. The retail sector was identified as the most important for the project's exploitation by the exploitation strategy report. They evaluated the system with regard to all the criteria. The system administrators were experienced in setting up and maintaining information systems in the business software sector. They evaluated the system only with regard to the criteria C3 and C4. Also, experienced independent scientists in the economic (as consultants) and computer science (as system administrators) fields working at another MARKET-MINER project partner (Informatics and Telematics Institute, Greece) evaluated the application for the same criteria.

The Process of Evaluation of Software Products, also referred to as MEDE-PROS (Colombo and Guerra, 2002) was used for evaluating the MARKET-MINER system. MEDE-PROS is in use for over 15 years, continually evolving and it has been applied to more than 360 software products.

The results of the evaluation of the MARKET-MINER software prototype are presented in Table 6 and they have been characterized as "very satisfactory" by the

SingularLogic research and development software assessment unit. MARKET-MINER has been decreed as worthy for recommendation for commercialization and addition to the Firm's software products suite.

Table 6. MARKET-MINER evaluation results. The rows with white background are those of the consultants, while those with grey background represent the evaluation of the system administrators.

Criterion	Criterion Performance	Criterion Importance
C1	86%	0,78
C2	83%	0,88
C3	91%	0,88
C4	83%	0,64
C3	86%	0,92
C4	61%	0,92

Chapter 5

Metamodels and Model Transformations

This chapter aims to show the technical details for implementing the model transformations that were introduced in §Chapter 4 and that occur in the ASEME process. Model transformations allow the *models of a previous phase to be automatically transformed to models of a next phase*. The requirement for defining a model transformation procedure is the existence of metamodels that describe the source and target models.

There are four types of transformation techniques (Langlois et al., 2007) each of which is handled by different technologies:

- **Model to Model (M2M)** transformation. This kind of transformation is used for transforming a type of graphical model to another type of graphical model. An example of such transformation is the SAG2SUC transformation where a System Actors Goals (SAG) diagram is transformed to a System Use Case (SUC) diagram. 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. A liveness formula proposes such a kind of syntax. The graphical model must have a metamodel. Then, a transformation of the text to a graphical model can be defined, as in the case of the SRM2IAC transformation.

- **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.

In the heart of the model transformation procedure is the Eclipse Modeling Framework (EMF), as it was presented by Budinsky et al. (2003). EMF unifies Java, XML, and UML technologies, allowing the modeler to switch between them as they provide the same information in a different representation. Figure 78 shows how EMF unifies these three. Regardless of which one is used to define it, an EMF model is the common high-level representation that "glues" them all together.

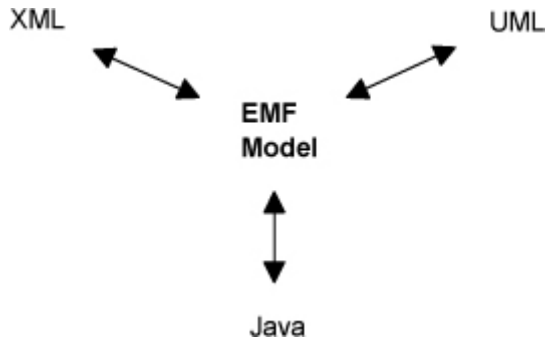


Figure 78. The EMF model unifies Java, XML, and UML technologies (Budinsky et al., 2003).

Ecore (Budinsky et al., 2003) is EMF's model of a model (metamodel). Using ecore, a modeler can define models. 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). Figure 79 shows the ecore metamodel in detail.

A similar technology, the Meta-Object Facility (MOF), is an Object Management Group (2001) standard for representing metamodels and manipulating them. There are a number of essential concepts used in MOF modeling. A Package is used to encapsulate a collection of related Classes and Associations. Packages can also contain simple type definitions. Classes exist in the commonly-used sense of the word, describing an object and its properties. These properties are represented through Attributes and References, which can be inherited using a multiple-inheritance system. Attributes have a name and a type. This includes a range of

types from basic types such as integers, strings, and booleans to more complex types such as enumerations, and through to structured types. In addition, attributes have both upper and lower limits on the number of times that they can appear within a class instance. An Association is used to represent a relationship between instances of two classes, each of which plays a role within the association. Associations can have the additional property of containment; an association represents a containment relationship if one of the participant classes does not exist outside the scope of the other. A Class participating in an association can also contain a Reference to the association. A reference appears much like an attribute, but reflects the set of class instances that participate in the Association with the containing class instance.

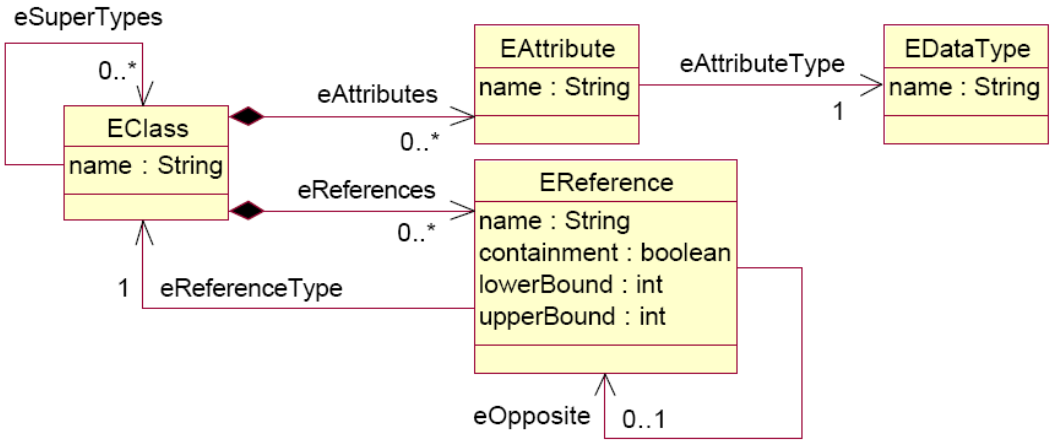


Figure 79. The Ecore metamodel (Budinsky et al., 2003).

MOF is older than EMF and it influenced its design. MOF was initially designed primarily for use with the Common Object Request Broker Architecture (CORBA). CORBA is an architecture that enables programs, called objects, to communicate with one another regardless of what programming language they were written in or what operating system they're running on.

EMF, on the other hand, is a product of the Eclipse project, an open source project and was intended as a low-cost tool to obtain the benefits of formal modeling and Java code generation. As a consequence, one could say that EMF took a bottom-up approach whereas MOF took a top-down approach (Gerber and Raymond, 2003).

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. EMF is used today by the IBM WebSphere/Rational product family, other Independent Software Vendors (ISVs) like TogetherSoft, Ensemble, Versata and Omondo and a large open source community becoming a de facto standard in MDE. Moreover, third parties define MDE tools based on EMF technology, like the

openArchitectureWare (oAW) platform for model-driven software development. For all these reasons it was decided that for the ASEME metamodeling, the EMF technology would be used.

In the following, the AMOLA metamodels are presented in detail. Then, the automatic model transformations that occur in the ASEME process are described along with the utilized technologies. Finally, the overall tool supported MDE ASEME process is summarized in §5.3.

5.1 The Metamodels

Bernon et al. (2005) present the metamodels of the most popular AOSE methodologies. These include metamodels for the Gaia and Tropos methodologies. These models were used as a basis for defining the SAG and SRM metamodels. The SUC and IAC metamodels were mainly inspired by the UML metamodel. Existing metamodels can be found in ecore format (and a number of other formats as well) in the zoos repository of the AtlanMod (for "Atlantic Modeling") team (2008), located in Nantes (France).

The following paragraphs define the AMOLA metamodels. Each time, the metamodel that inspired the AMOLA metamodel is presented as well. The goal of the author in doing so is twofold. On one hand the reader can see for himself that models following the original metamodel can be transformed to AMOLA models with a straightforward process (achieved just by aligning concepts), thus such method fragments can be easily inserted in the AMOLA process. On the other hand, the simplicity of the AMOLA models will become evident, showing that it is ideal for agile development. The metamodels are presented in figures so that they are better understood by the reader. They are, however, also included in their ecore XML format in Annex 3. The graphical models are automatically generated using the Eclipse EMF tools.

5.1.1 System Actor Goal model (SAG)

The SAG model is a subset of the Actor model of the Tropos ecore model (Actor Concept 1.0) as it is maintained in the zoos repository. It is the same metamodel that appears in Susi et al. (2005). It is presented graphically in Figure 80.

As the AMOLA System Actors Goals diagram does not use all these concepts a more compact version of the Tropos Actor Concept metamodel has been defined, the one that appears in Figure 81. Thus, there are 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* (according to §3.2.1). The reader should notice the choice to add the *requirements EAttribute* of *Goal* where the

"requirements per goal" (RPG) information is stored. The SAG metamodel is presented in ecore XML format in Listing 20 in Annex 3.

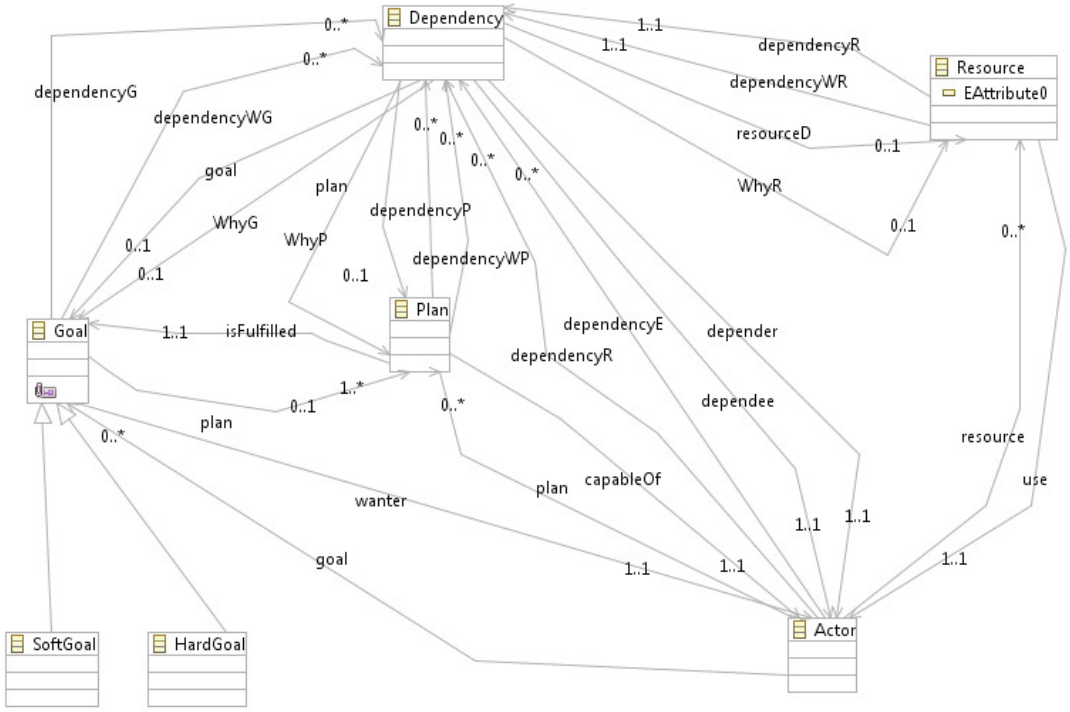


Figure 80. The Tropos Actor Concept metamodel (from AtlanMod repository).

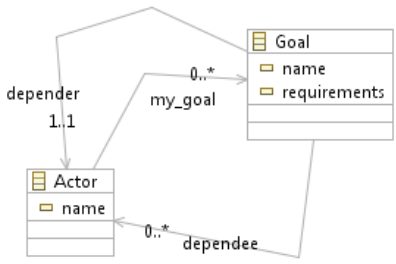


Figure 81. The AMOLA SAG metamodel

5.1.2 Use case model (SUC)

The same procedure was followed for defining the SUC metamodel. The UML metamodel is a huge metamodel but a fragment of it including the concepts related to use case diagrams is presented in Figure 82. The fragment is available as-is in the AtlanMod (2008) zoos site.

Again, a compact metamodel containing the concepts used by AMOLA has been defined using the ecore metametamodel and it is presented in Figure 83. It follows the definitions using free text of §3.3.1.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). The SUC metamodel is presented in ecore XML format in Listing 21 in Annex 3.

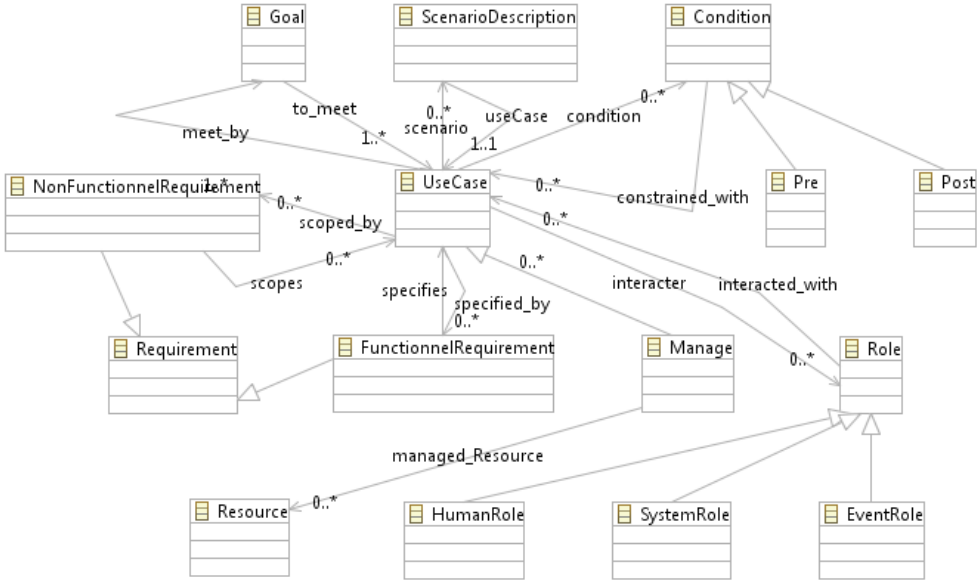


Figure 82. The Usecase fragment of the UML metamodel (from AtlanMod repository).

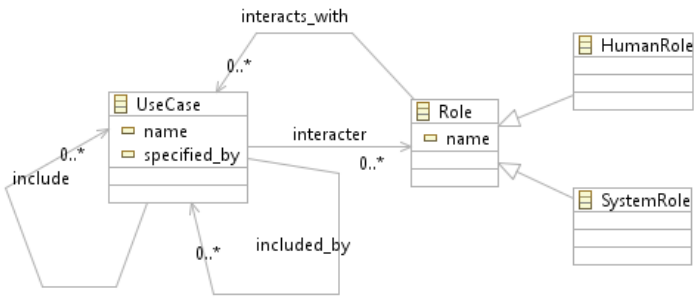


Figure 83. The AMOLA SUC metamodel

5.1.3 Role model (SRM)

The SRM model is a subset of the GAIA 1.0 ecore model as it is archived in the zoos repository of the AtlanMod (2008). It is the same metamodel that appears in Bernon et al. (2005). It is presented graphically in Figure 84.

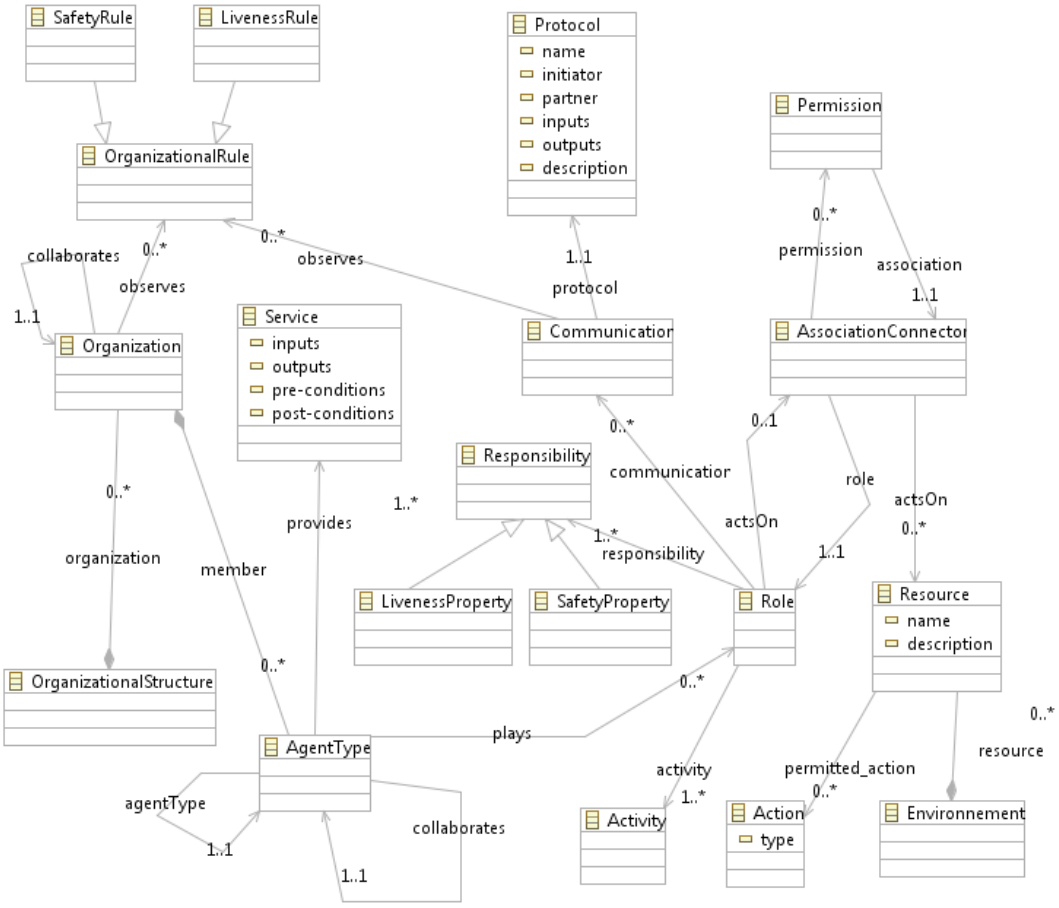


Figure 84. The GAIA metamodel (from AtlanMod repository).

A more compact metamodel containing the concepts used by AMOLA (according to §3.3.3) has been defined using the ecore metametamodel and it is presented in Figure 85. 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's attributes name and participant refer to the relevant items in the EAC model.

The *Role* concept also has the *name* and *liveness* attributes (the first is the role name and the second its liveness formula). The SRM metamodel is presented in ecore XML format in Listing 22 in Annex 3. The reader should note that the *functionality* attribute of the *Activity* concept incorporates the information that is included in the AMOLA Functionality Table (associate each activity to a unique functionality).

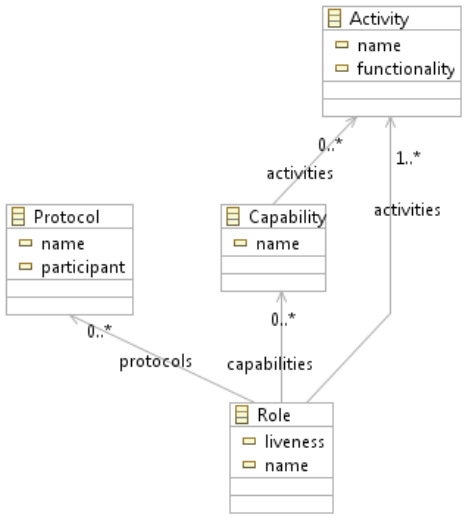


Figure 85. The AMOLA SRM metamodel.

5.1.4 Intra-agent control model (IAC)

The inspiration for defining the IAC metamodel mainly came from the UML statechart definition (presented in Figure 86).

Aiming to define the statechart using the AMOLA definition of statechart (§3.4.1), the IAC metamodel differs significantly from the UML statechart (see Figure 87). 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).

Thus, the IAC metamodel contains nodes and transitions according to Definition 3.7 in §3.4.1. The metamodel 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 (see a sample namespace for the meetings management system in the next section with the transformations).

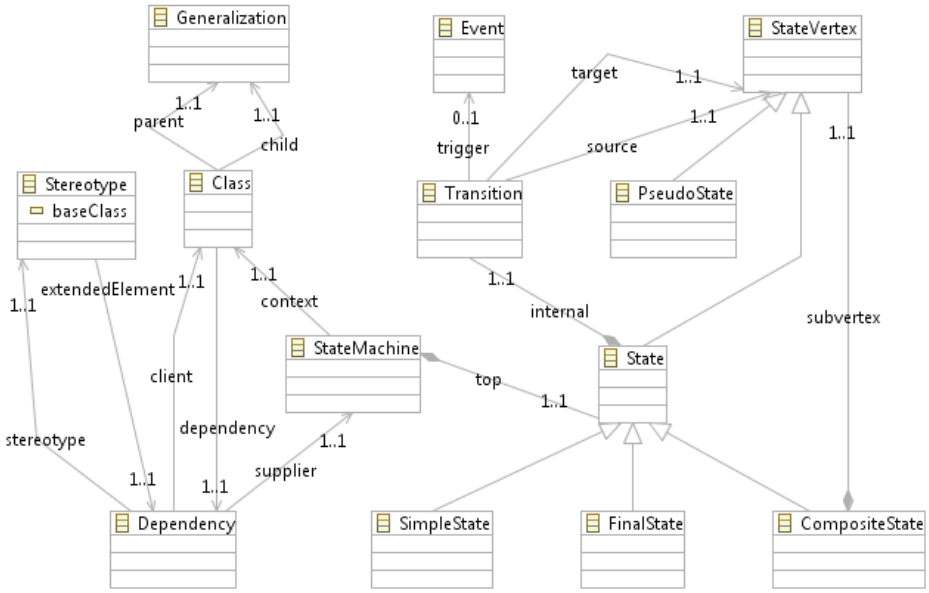


Figure 86. The Statecharts fragment of the UML metamodel (from AtlanMod repository).

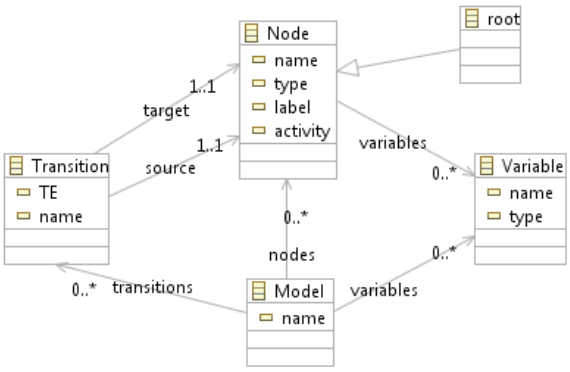


Figure 87. The AMOLA IAC metamodel.

The nodes contain the following attributes (followed by the relevant concept name in the statechart definition):

- *name* (Name). The name of the node,
- *type* (λ). The type of the node (one of AND, OR, BASIC, START, END),
- *label* (label). The node's label, and
- *activity* (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”). Finally the transitions have four attributes:

- *name*, usually in the form <source node label>TO<target node label>
- *TE*, the transition expression following the relevant grammar (see Listing 2)
- *source*, the source node, and,
- *target*, the target node.

The IAC metamodel is presented in ecore XML format in Listing 23 in Annex 3.

5.2 The Transformations

Three types of transformation techniques are used in the ASEME process. Firstly, two M2M transformations occur, SAG2SUC (read SAG to SUC) and SUC2SRM, then a T2M, the SRM2IAC and, finally, a M2T, the IAC2JADE. Each type of transformation required a different technology use, including programming language, tools usage and expertise in building a relevant project. However, now that the tools have been created in Eclipse the ASEME developer can achieve the transformations with a single “click”. This section describes how the transformation tools were realized in the context of this thesis.

5.2.1 M2M Transformations

For model to model transformation the Atlas Transformation Language was used (ATL). Another alternative to Atlas would be the Query-View Transformation language, however, Atlas 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 (Jouault and Kurtev, 2006b) the decision was to choose the better documented one.

The structure of an ATL transformation project in the Eclipse Integrated Development Environment (IDE) is shown in Figure 88. Three folders contain the files used, the *ATL_files* folder containing the transformation scripts in ATL, the *metamodels* folder containing the metamodels and the *models* folder containing the models.

5.2.1.1 SAG2SUC transformation

The SAG2SUC transformation is a classic M2M transformation using ATL. The ATL rules are presented in Figure 88. At the top, 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 *interacter* references of the SUC UseCase. The ATL engine realizes that the transformation is not about an EAttribute (like in the case of the *name* attribute) and 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.

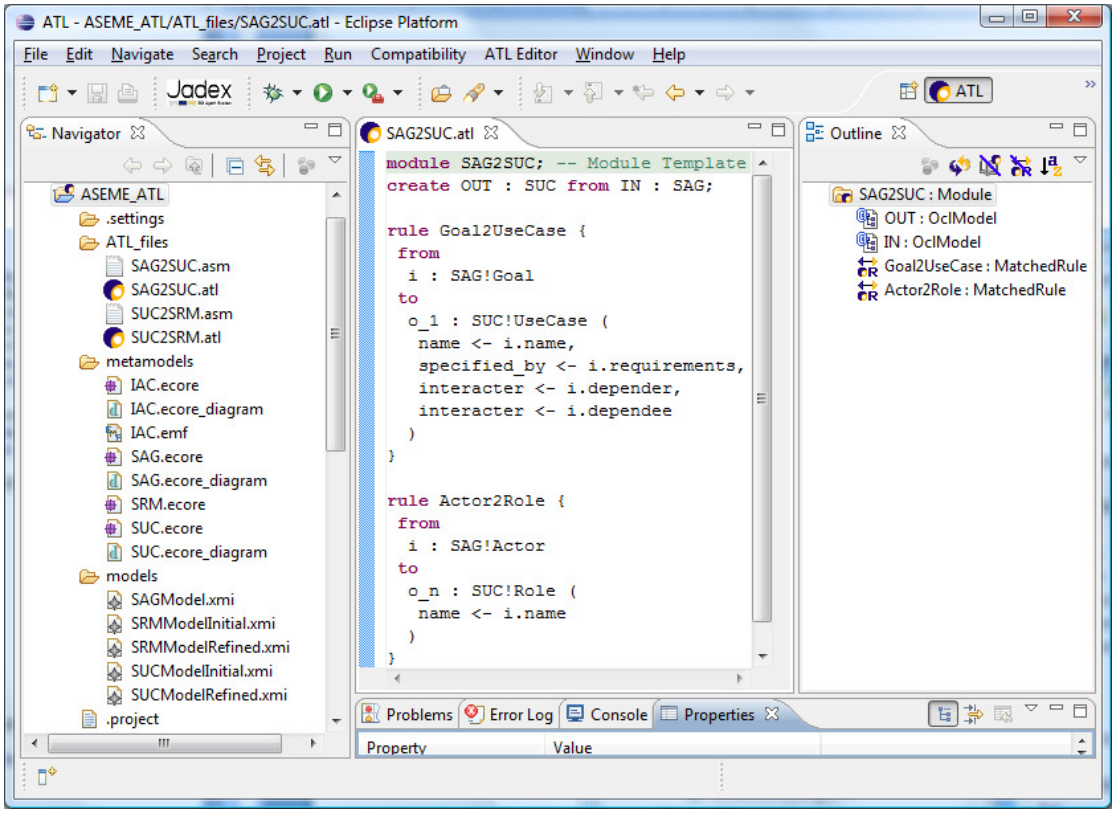


Figure 88. The eclipse ATL project for the SAG2SUC and the SUC2SRM M2M transformations.

Thus, having the SAG2SUC.atl file the requirements for achieving a M2M transformation are met and the general scheme presented in Figure 16 can be instantiated to the one presented in Figure 89.

To execute the transformation the ATL Transformation type in the Eclipse Run Configurations must be instantiated. It launches a dialog where the type of metamodels (EMF in this case) is chosen along with the metamodel and model files (see Figure 90). The source and target models for the meetings management example are presented in Listing 34 (SAGModel.xmi) and Listing 35

(SUCModelInitial.xmi) in Annex 6 so that the reader can see the input and output models of the transformation. Thus, if the ASEME modeler uses the same names for his own models he can automatically cause the transformation by executing the ASEME_SAG2SUC transformation. This holds for every transformation defined in this chapter.

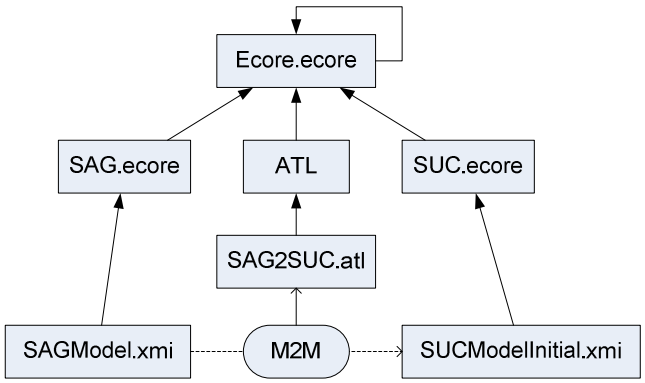


Figure 89. The SAG2SUC M2M transformation scheme.

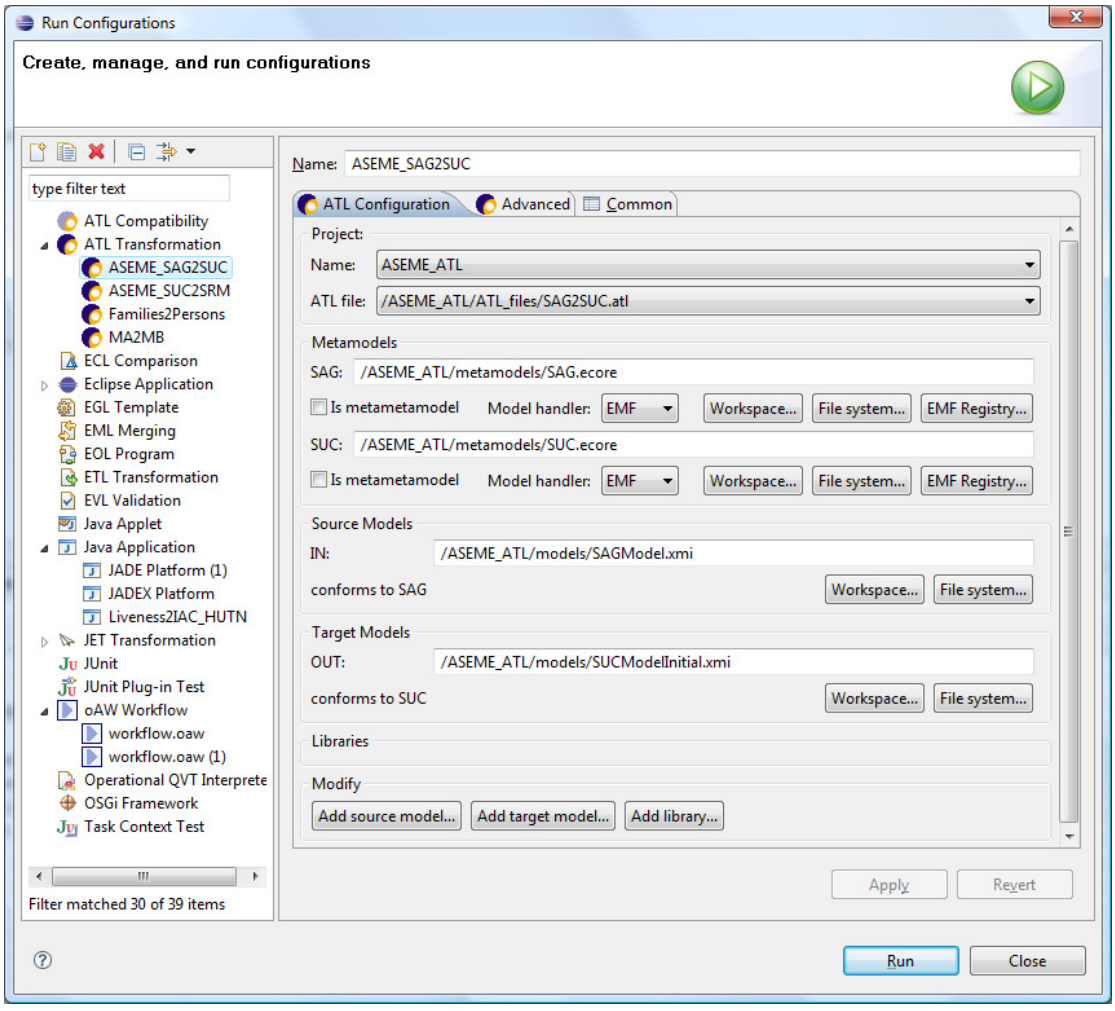


Figure 90. The ATL Transformation Run Configuration of Eclipse.

5.2.1.2 SUC2SRM transformation

After the SAG2SUC transformation the modeler works on the SUC model (using the *Sample Reflective Ecore Model Editor* of Eclipse) adding detail in the form of *included* use cases, assigning specific technologies to the defined use cases and cardinality. When he is finished he can proceed to the next transformation, the SUC2SRM one.

The ATL technology is again used and the rules for this transformation are presented in Listing 6. It has one more rule as the UseCases can be transformed to either Capabilities (if they *include* others) or to simple activities. This time, again, the first rule (Role2Role) orchestrates the transformation procedure.

The transformation scheme is presented in Figure 91. The ATL rules are applied to the refined SUC model and create the initial SRM model. The source and target models for the meetings management example are presented in Listing 36 (SUCModelRefined.xmi) and Listing 37 (SRMModelInitial.xmi) in Annex 6. The ASEME modeler just has to execute the ASEME_SUC2SRM ATL transformation.

Listing 6. The SUC2SRM ATL Transformation script (SUC2SRM.atl file).

```
-- @path SRM=/ASEME_ATL/metamodels/SRM.ecore
-- @path SUC=/ASEME_ATL/metamodels/SUC.ecore

module SUC2SRM; -- Module Template
create OUT : SRM from IN : SUC;

rule Role2Role {
  from
    i : SUC!Role
  to
    o_1 : SRM!Role (
      name <- i.name,
      activities <- Sequence {} ->
                  union(i.interacts_with -> select(e | not(e.interacter->size()>1))),
      capabilities <- Sequence {} ->
                  union(i.interacts_with -> select(e | e.interacter->size()>1))
    )
}

rule UseCase2Activity {
  from
    i : SUC!UseCase ((not(i.interacter->size()>1))and(i.include->size()=0))
  to
    o_1 : SRM!Activity (
      name <- i.name,
      functionality <- i.specified_by
    )
}

rule UseCase2Capability {
  from
    i : SUC!UseCase ((i.interacter->size()>1)or(i.include->size()>0))
  to
    o_1 : SRM!Capability (
      name <- i.name,
      activities <- i.include
    )
}
```

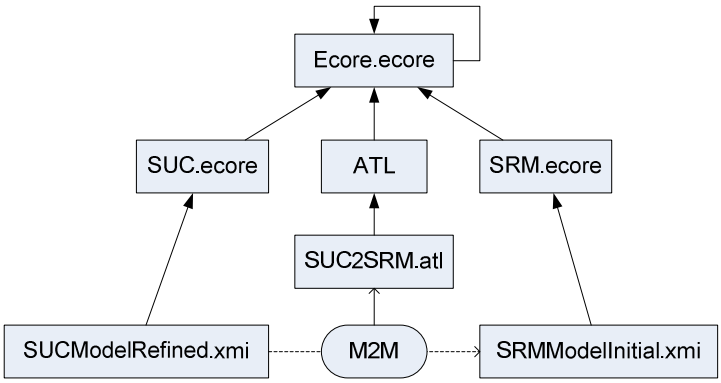


Figure 91. The SUC2SRM M2M transformation scheme.

5.2.2 T2M – The SRM2IAC transformation

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 (2006) presented the xtext framework in the context of the oAW project. 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 (comments, name, attributes and references). 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. (2008) described an implementation of the Human-Usable Textual Notation (HUTN) specification of OMG (Object Management Group, 2004) using Epsilon, the Extensible Platform for Specification of Integrated Languages for mOdel maNagement (Kolovos et al., 2006), which is a suite of tools for MDE. OMG created HUTN aiming to offer three main benefits to MDE:

- a generic specification that can provide a concrete HUTN language for any MOF model
- the HUTN languages to be fully automated both for production and parsing
- the HUTN languages to conform to human-usability criteria

An example of a HUTN-generated language is presented in Listing 7 (the text in bold face is the literal text stream, whereas the text in bold-and-italic face describe omitted detail which should not be taken literally).

Listing 7. A HUTN generated language example

```
FamilyPackage "id-001" {
  Class instances here
  Association instances or links here
}
```

HUTN mappings are defined using EBNF rules (see Listing 8). The italicized words enclosed in angle brackets indicate a placeholder for a literal Family Package value that must be substituted with an actual value (e.g., <*PackageName*>). The HUTN example in Listing 7 adheres to the grammar extract presented in Listing 8.

Listing 8. An extract of the EBNF rules for HUTN mappings (Object Management Group, 2004)

```
PackageInstance := PackageHeader { PackageBody }
PackageHeader := <PackageName> PackageIdentifier
...
```

The HUTN implementation architecture of the Epsilon platform is presented in Figure 92, where the reader can see the text model being transformed to an abstract syntax tree (AST) model that is meta-model agnostic (does not depend to a meta-model or modeling language). It can then be transformed to an intermediate model where references between elements are resolved and which then can be validated with regard to the HUTN specification. Then a second transformation creates the target model which may be defined in EMF, MOF, or other modeling languages.

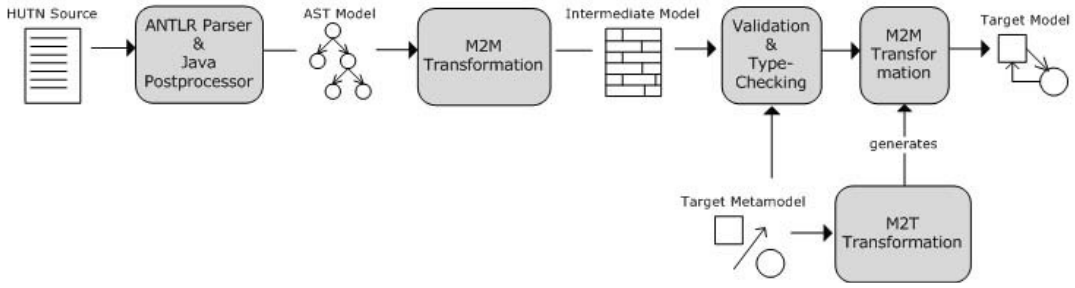


Figure 92. The HUTN implementation architecture (Rose et al., 2008)

The HUTN implementation automates the transformation process by eliminating the need for a grammar specification by auto defining it accepting as input the relevant EMF meta-model. This is the main reason for choosing HUTN for ASEME. In Figure 93 the eclipse project for the SRM2IAC transformation is presented. It is a simple Java

project where after installing the epsilon project, the HUTN nature (by right-clicking on the project icon on the Package explorer) has been turned on. The input for this transformation is the SRM refined model, mainly the liveness formula.

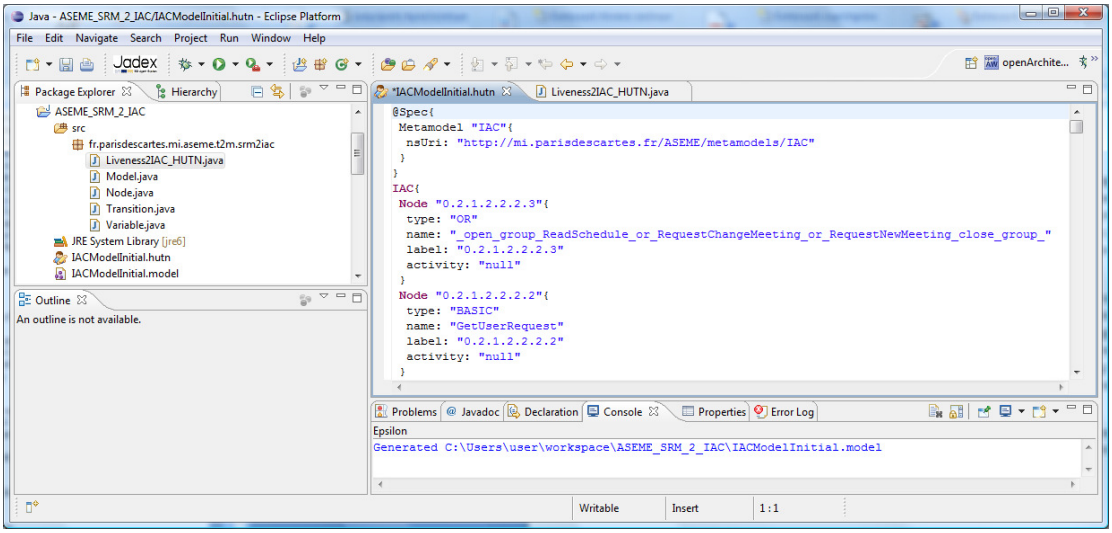


Figure 93. The Eclipse project for T2M transformation.

Then, the transformation process from a liveness formula to a statechart (IAC model) presented in §4.5 (see Listing 3) is implemented in the "Liveness2HUTN.java" file (presented in Listing 24 in Annex 4). As its name suggests, this Java program transforms the liveness formula of an SRM role to a HUTN file. This transformation is a T2T transformation, however, there is no specific MDE technique used for this (just a Java string manipulation function). The algorithm creates a model including nodes, transitions and variables objects by instantiating the relevant classes Model, Node, Transition and Variable each of which has a method toHutnString() which prints the element data in the format shown in Figure 93. The usage of the HUTN technology also helped a lot in debugging the algorithm as the output was in human-readable format.

The ASEME modeler just has to execute the "Liveness2HUTN.java" file in order to create the HUTN representation of the IAC model. The .hutn file created for the meetings management project from the liveness formula of the refined SRM model is presented in Listing 39 (IACModelInitial.hutn) in Annex 6. Then, simply by right-clicking to the hutn file and selecting to generate the model the initial IAC model is generated (shown in XML format) in Listing 40 (IACModelInitial.model). An extract of this file where the XML elements representing the Hutn representation part visible in Figure 93 is presented in Listing 9.

Thus, the IAC model has now been initialized with the information available in the SRM model and it can be refined in the design phase, again using the *Sample Reflective Ecore Model Editor* of Eclipse.

Listing 9. An extract from the IACModelInitial.model file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:IAC="http://mi.parisdescartes.fr/ASEME/metamodels/IAC">
  <IAC:Node name="_open_group_ReadSchedule_or_RequestChangeMeeting_or_
RequestNewMeeting_close_group_" type="OR" label="0.2.1.2.2.2.3" activity="null" />
  <IAC:Node name="GetUserRequest" type="BASIC" label="0.2.1.2.2.2.2" activity="null"
/>
...
```

5.2.3 M2T Transformation

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. Klatt (2007) takes a closer look in M2T transformation and describes the usage of the *Xpand* language in the oAW project. The Xpand language allows to define templates and to access functions defined in the *Xtend* language (see more information below).

Another commonly used M2T transformation language is the Java Emitter Templates (JET). It provides a framework and several facilities for code generation. JSP[7] like templates are used and by this it makes it easy to learn for developers already familiar with this technology. It is easy to extend with custom tags like it is possible for JSPs.

The advantages of Xpand are the fact that it is source model independent, which means that any of the oAW project 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, oAW allows for defining workflows that can help a modeler to achieve multiple parsings of the model with different goals. These are the reasons for choosing Xpand for the ASEME M2T transformation.

An overview of the JADE framework, for which the target text model will be generated is presented before the description of the transformation process.

5.2.3.1 The Java Agent Development Framework (JADE)

JADE (Bellifemine et al, 2007) is a software development framework fully implemented in Java language aiming at the development of multi-agent systems and applications that comply with FIPA standards for intelligent agents. JADE provides standard agent technologies and offers to the developer a number of features in order to simplify the development process:

[7] JavaServer Pages (JSP) technology provides a simplified, fast way to create dynamic web content. Find more information in http://java.sun.com/products/jsp/

- Distributed agent platform. The agent platform can be distributed on several hosts, each of which executes one Java Virtual Machine.

- FIPA-Compliant agent platform, which includes the Agent Management System the Directory Facilitator and the Agent Communication Channel (FIPA TC Agent Management, 2002).

- Efficient transport of agent communication language (ACL) messages between agents (FIPA TC Communication, 2002).

All inter-agent communication is performed through message passing and the FIPA ACL is the language that is used to represent the messages. Each agent is equipped with an incoming message box and message polling can be blocking or non-blocking with an optional timeout. Moreover, JADE provides methods for message filtering. The developer can apply advanced filters on the various fields of the incoming messages such as sender, performative or ontology.

FIPA specifies a set of standard interaction protocols such as FIPA-request, FIPA-query, etc. that can be used as standard templates to build agent conversations. For every conversation among agents, JADE distinguishes the role of the agent that starts the conversation (initiator) and the role of the agent that engages in a conversation started by another agent (responder). According to the structure of these protocols, the initiator sends a message and the responder can subsequently reply by sending a not-understood or a refuse message indicating the inability to achieve the rational effect of the communicative act, or an agree message indicating the agreement to perform the communicative act. When the responder performs the action he must send an inform message. A failure message indicates that the action was not successful. JADE provides ready-made behaviour classes for both roles, following most of the FIPA specified interaction protocols (FIPA TC Communication, 2002). JADE provides the *AchieveREInitiator* and *AchieveREResponder* classes, a single homogeneous implementation of interaction protocols such as these mentioned above. Both classes provide methods for handling all possible protocol states.

In JADE, agent tasks or agent intentions are implemented through the use of behaviours. Behaviours are logical execution threads that can be composed in various ways to achieve complex execution patterns and can be initialized, suspended and spawned at any given time. The agent core keeps a task list that contains the active behaviours. JADE suggests the use of one thread per agent instead of one thread per behaviour to limit the number of threads running in the agent platform. A scheduler, hidden to the developer, carries out a round robin policy among all behaviours available in the queue. The behaviour can release the execution control with the use of blocking mechanisms, or it can permanently remove itself from the queue in run time. Each behaviour performs its designated operation by executing the core method *action()*.

Behaviour is the root class of the behaviour hierarchy that defines several core methods and sets the basis for behaviour scheduling as it allows state transitions

(starting, blocking and restarting). The children of this base class are *SimpleBehaviour* and *CompositeBehaviour*.

The classes that descend from *SimpleBehaviour* represent atomic simple tasks that can be executed a number of times specified by the developer. The class *CyclicBehaviour* models atomic behaviours that must be executed forever. So, its *done()* method always returns false.

Classes descending from *CompositeBehaviour* support the handling of multiple behaviours according to a policy. The actual agent tasks that are executed through this behaviour are not defined in the behaviour itself, but inside its children behaviours. The class *SequentialBehaviour* is a *CompositeBehaviour* that executes its sub-behaviours sequentially and terminates when all sub-behaviours are done. The class *ParallelBehaviour* is a *CompositeBehaviour* that executes its sub-behaviours concurrently and terminates when a particular condition on its sub-behaviours is met. Proper constants to be indicated in the constructor of this class are provided to create a ParallelBehaviour that ends when all its sub-behaviours are done, when any one among its sub-behaviour terminates or when a user defined number N of its sub-behaviours have finished.

The developer creates his agents by extending the JADE *Agent* class. He can add any number of behaviours along with defining the agent's initialization and termination handling functionality. A special descendant of the *Agent* class, the *GUIAgent*, allows for the creation of agents with a graphical user interface (GUI), allowing for the agent's interaction with a human user. The latter is facilitated by a GUI event exchange mechanism that also allows the definition of parameters that accompany the event. Whenever a specified GUI event occurs the agent can add a new behaviour passing to its constructor the relevant parameters and a reference to the GUI so that the behaviour can reply to the user.

5.2.3.2 The IAC2JADE Transformation

The eclipse M2T project (IAC_EMF.generator) and its referenced projects are presented in Figure 94. It references the IAC_EMF.edit and IAC_EMF.editor projects. They are automatically generated by the IAC_EMF project that is an eclipse EMF project. There, the IAC.genmodel (automatically generated from IAC.ecore) is used to generate the edit and editor projects that contain the automatically generated java classes for editing IAC models.

Figure 94 shows on the right side the refined IAC model for the meetings management system. The first element in the model is the *Model* concept whose *name* attribute, which is shown next to its type (Model), is the namespace of the project (i.e. *fr.parisdescartes.mi.meetingsmanagement*).

The ASEME IAC2JADE transformation project is the IAC_EMF.generator (or simply generator project). It has two main folders, the scr and the src-gen. The first one contains the transformation templates and helper files, while the second is the one that receives the generated java files. The transformation process is comprised of multiple steps and the workflow file of oAW allows to define this process (another

advantage of using Xpand). The workflow file can be used to define execution parameters, usually through property files, and file generating components.

The workflow file used in this process is presented in Listing 10. It initially loads some parameters through a property file (the “workflow.properties” file is presented in Listing 11 in detail), specifically the name of the IAC model file, the character encoding used and the directory for producing the source code. Then, a component (the one with with id *xmiParser*) parses the model file and validates it. A following component cleans up the src-gen folder. Then, the next component generates a file. The goal of this pass is to get the name of the model which corresponds to its namespace. The “path.properties” file is generated that contains this name. This component executes the Preprocessing template shown in Listing 12.

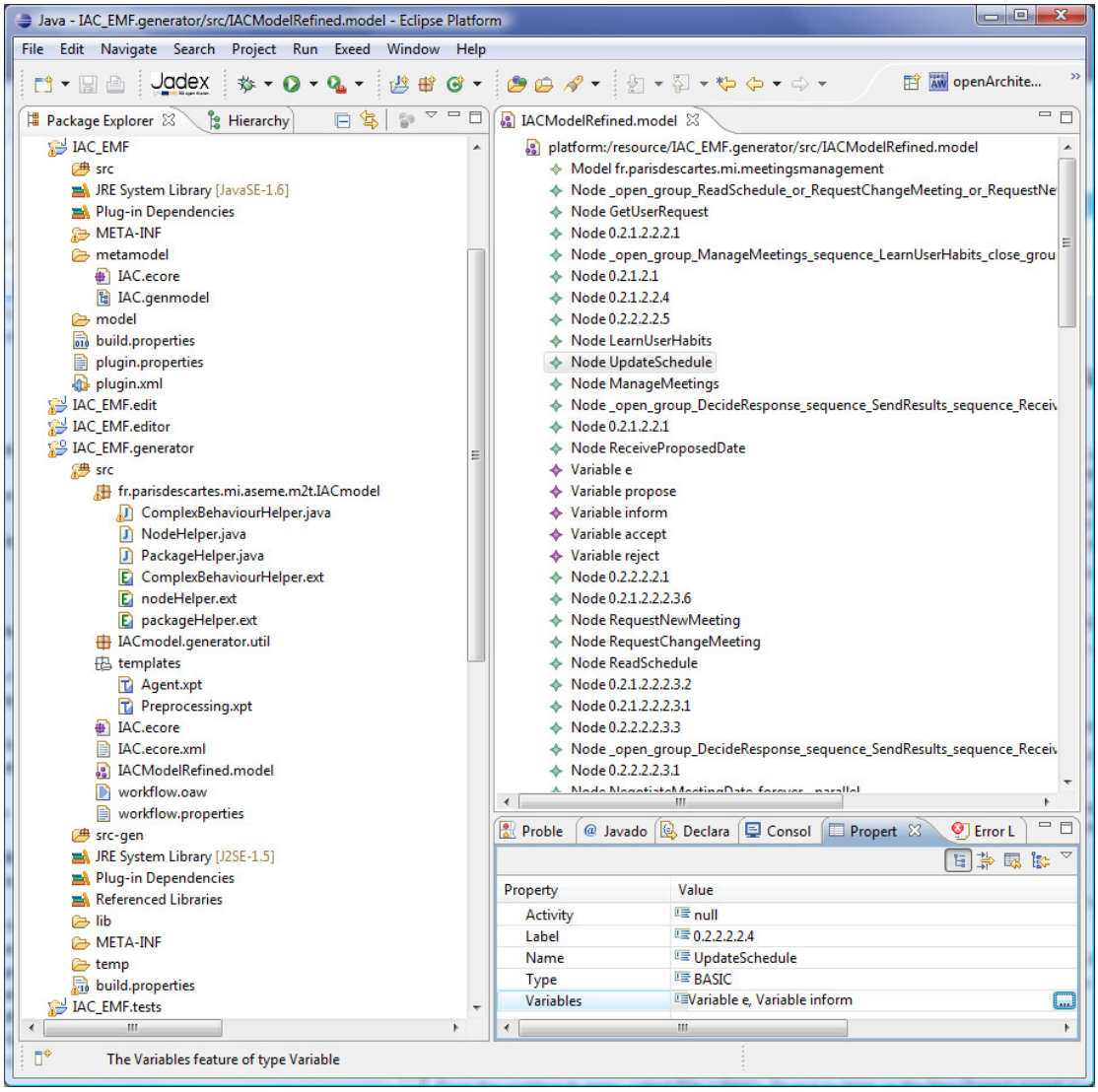


Figure 94. The JADE code generator project with its prerequisite projects in eclipse.

Listing 10. The workflow definition for the IAC2JADE transformation (workflow.oaw).

```
<workflow>

  <property file="workflow.properties"/>

  <component id="xmiParser" class="org.openarchitectureware.emf.XmiReader">
      <modelFile value="${modelFile}"/>
      <metaModelPackage value="IAC.IACPackage"/>
      <outputSlot value="model"/>
      <firstElementOnly value="true"/>
  </component>

  <component id="dirCleaner"
class="org.openarchitectureware.workflow.common.DirectoryCleaner">
    <directory value="${srcGenPath}"/>
  </component>

  <component id="path_generator" class="org.openarchitectureware.xpand2.Generator">
      <metaModel id="mm" class="org.openarchitectureware.type.emf.EmfMetaModel">
        <metaModelPackage value="IAC.IACPackage"/>
      </metaModel>
      <expand value="templates::Preprocessing::pathProperty FOR model"/>
      <outlet path="${srcGenPath}/" overwrite='true'/>
  </component>

  <property file="${srcGenPath}/path.properties"/>

  <component id="code_generator" class="org.openarchitectureware.xpand2.Generator">
    <metaModel id="mm" class="org.openarchitectureware.type.emf.EmfMetaModel">
      <metaModelPackage value="IAC.IACPackage"/>
    </metaModel>
    <expand value="templates::Agent::javaClass FOR model"/>
    <outlet path="${srcGenPath}/${path}/">
      <postprocessor class="org.openarchitectureware.xpand2.output.JavaBeautifier"/>
    </outlet>
  </component>

</workflow>
```

Listing 11. The workflow properties file (workflow.properties)

```
modelFile=IACModelRefined.model
srcGenPath=src-gen
fileEncoding=UTF-8
```

Listing 12. The preprocessing xpand template file (Preprocessing.xpt)

```
«IMPORT IAC»
«EXTENSION fr::parisdescartes::mi::aseme::m2t::IACmodel::packageHelper»

«DEFINE pathProperty FOR IAC::Model»
   «FILE "path.properties"»
path=«packagePath()»
   «ENDFILE»
«ENDDEFINE»
```

The preprocessing Xpand template file imports the IAC metamodel namespace and uses the packageHelper Xtend helper file shown in Listing 13. As the reader can see

the helper file just declares a function that can then be implemented by a programming language, in this case Java. The full namespace of the Java class and called method is included in the helper file method definition. The Java implementation is shown in Listing 14; it gets the name of the processed IAC model and replaces the dots with slashes so as to produce the folder for storing the generated java files. In the case of the meetings management project for the package namespace "fr.parisdescartes.mi.meetingsmanagement" the output of this method is the "fr/parisdescartes/mi/meetingsmanagement".

Listing 13. The packageHelper xtend file (PackageHelper.ext)

```
import IAC;

String packagePath( Model e ) :
   JAVA fr.parisdescartes.mi.aseme.m2t.IACmodel.PackageHelper.packagePath(IAC.Model);
```

Listing 14. The packageHelper Java implementation class (PackageHelper.java)

```
package fr.parisdescartes.mi.aseme.m2t.IACmodel;

import IAC.Model;

public class PackageHelper {

  public static String packagePath( Model e ) {
    System.out.print("defining the package path");
    String result =e.getName().replaceAll("\\.", "/");
    return result;
  }
}
```

Returning to the "Preprocessing.xpt" file (shown in Listing 12), after the IMPORT and EXTENSION statements there is the DEFINE statement that defines templates. The specific define template is named *pathProperty* (this is the template that the workflow file invoked) and is executed for an IAC model. The FILE statement defines the name of the file that is outputted and its body is the file template. In this case the file contains a simple line that writes "path=" and then the result of the *packagePath* Xtend function is placed. This is how Xpand cooperates with Xtend.

The workflow continues by loading the "path.properties" created property file and then reloads the IAC model for producing the java classes in the path defined by the path property. The agent xpand template file ("Agent.xpt", see Listing 29 in Annex 5) is a much more complex Xpand template file, the same holds for its Xtend helper functions ("nodeHelper.ext" and "ComplexBehaviourHelper.ext", Listing 30 and Listing 32 in Annex 5) and their java implementation files ("NodeHelper.java" and "ComplexBehaviourHelper.java", Listing 31 and Listing 33 in Annex 5). An extract of the agent Xpand file is shown in Listing 15 so as to explain its functionality.

Listing 15. An extract from the agent xpand template file (1)

```
«IMPORT IAC»
«EXTENSION fr::parisdescartes::mi::aseme::m2t::IACmodel::nodeHelper»
«EXTENSION fr::parisdescartes::mi::aseme::m2t::IACmodel::ComplexBehaviourHelper»

«DEFINE javaClass FOR IAC::Model»
  «LET name AS packageName»
    «EXPAND nodeClass(packageName, this) FOREACH nodes»
    «EXPAND variableHolderClass(packageName, this) FOREACH variables»
  «ENDLET»
«ENDDEFINE»

«DEFINE variableHolderClass(String packageName, Model model) FOR IAC::Variable»
  «FILE variableHolderFileName()»
  package «packageName»;
  import jade.core.behaviours.Behaviour;
  «IF type.compareTo("ACLMessage")==0»import jade.lang.acl.ACLMessage;«ENDIF»

  public class «type»Holder {
    «type» «lowerCaseFirstCharacterOfVariable(this)» = null;
    Behaviour owner;

    public «type»Holder(Behaviour owner) {
      super();
      this.owner = owner;
    }

    public «type» get«type»() {
      return «lowerCaseFirstCharacterOfVariable(this)»;
    }

    public void set«type»(«type» «lowerCaseFirstCharacterOfVariable(this)») {
      this.«lowerCaseFirstCharacterOfVariable(this)» =
«lowerCaseFirstCharacterOfVariable(this)»;
    }

    public Behaviour getOwner() {
      return owner;
    }
  }
  «ENDFILE»
«ENDDEFINE»
...
```

The first definition (*javaClass*), the one invoked by the workflow file, takes an IAC model concept and expands its variables and nodes. It defines the *packageName* variable using the Xpand LET statement setting it to the model's *name* attribute.

For each variable in the model a java class will be created (through the *variableHolderClass* expansion definition). The package is defined by the *packageName* parameter. If the variable type is that of an ACLMessage then the relevant class is imported from the jade framework. For all other variable types it is assumed that the ontology created for this project will contain them.

In the case of the meetings management project, there are two variable types, the *Meeting* variable type refers to a class defined in the ontology of the project and the *ACLMessage* variable type (see Listing 16 and Listing 50 respectively in Annex 6). The reader should notice that the class generated by the Xpand template is named after the type of the variable including the string "Holder". Thus, the class generated for the *Meeting* variable type is the *MeetingHolder* class. The latter has two attributes, the *owner*, which is a reference to a JADE Behaviour class (where the behavior that instantiates this variable is inserted through the class constructor) and the *meeting*

attribute that references the *Meeting* class. This approach, which is transparent to the developer, allows a behaviour to change a variable value and this change to be visible to all behaviours that share this variable.

Listing 16. The generated file MeetingHolder.java

```
package fr.parisdescartes.mi.meetingsmanagement;
import jade.core.behaviours.Behaviour;

public class MeetingHolder {
  Meeting meeting = null;
  Behaviour owner;

  public MeetingHolder(Behaviour owner) {
    super();
    this.owner = owner;
  }

  public Meeting getMeeting() {
    return meeting;
  }

  public void setMeeting(Meeting meeting) {
    this.meeting = meeting;
  }

  public Behaviour getOwner() {
    return owner;
  }
}
```

The agent Xpand template file continues by defining relevant templates for the agent class (extending the jade.core.Agent class) and its behaviours. Four types of behaviours are automatically generated according to the transformation process. The transformation algorithm is presented in pseudocode in Listing 17. The algorithm reads the statechart model (IAC) and creates Java source code files using templates (as defined in the Xpand agent template file). The information from the statechart is included in the "< >" signs whenever needed.

Listing 17. The transformation process of nodes to java classes from the IAC model to the JADE platform (IAC2JADE) in pseudocode.

```
For each node in S
  If node is root then
    create file f = "<name(node)>Agent.java"
    defining "public class <name(node)>Agent extends jade.core.Agent"
  Else if λ(node)="BASIC"
    create file f = "<name(node)>Behaviour.java"
    defining "public class <name(node)>Behaviour extends SimpleBehaviour"
  Else if λ(node)="AND"
    create file f = "<name(node)>Behaviour.java"
    defining "public class <name(node)>Behaviour extends ParallelBehaviour"
  Else if sons(node).size() = 2 and ∃ transitionExpression x | (node.2, x, node.2) ∈ δ
    create file f = "<name(node)>Behaviour.java"
```

```
    defining “public class <name(node)>Behaviour extends CyclicBehaviour”
  Else if sons(node).size() = 3 and ∃ transitionExpression x | (node.2, x, node.2) ∈ δ
    create file f = ”<name(node)>Behaviour.java”
    defining “public class <name(node)>Behaviour extends SimpleBehaviour”
  Else if ∃x∈sons(node) | λ(x)=CONDITION
    If sons(node).size() = 4
      create file f = ”<name(node)>Behaviour.java”
      defining “public class <name(node)>Behaviour extends SimpleBehaviour”
    Else
      create file f = ”<name(node)>Behaviour.java”
      defining “public class <name(node)>Behaviour extends SequentialBehaviour”
    End if
  Else
    create file f = ”<name(node)>Behaviour.java”
    defining “public class <name(node)>Behaviour extends SequentialBehaviour”
  End if
End for
```

In plain words, the idea behing the transformation algorithm is that each node of the statechart (IAC) is processed. If it is the root, then it is transformed to an agent JADE class. In what follows, the main details of implementation that have been implemented are discussed for each java class type. The agent class setup method is defined adding the sub-behaviours, i.e. the sons of the node that are of type OR, AND or BASIC (called the *eligible nodes* from now on). Notice that the nodes of type START, END and CONDITION are not transformed to Behaviour classes; they are only used for determining the other nodes' transformation to some kind of behaviour. For each of the other (than the root) eligible nodes one of the following holds (searching from top to bottom):

- If the node's type is “BASIC” then it is transformed to a JADE SimpleBehaviour (it extends the jade.core.behaviours.SimpleBehaviour class).
    - If the node's name starts with “Send”, then add a reference to the JADE ACLMessage class and write code for sending a message depending on the events of the transitions that have this node as their source (a result of such a transformation is the SendResultsBehaviour that can be viewed in Listing 68 in Annex 6).
    - Else, if the node's name starts with “Receive”, then add a reference to the JADE ACLMessage class and write code for receiving a message depending on the events of the transitions that have this node as their target. Also, add a reference to the MessageTemplate JADE class that is used for defining the type of message expected and instantiate it according to the events of the transitions that have this node as their target (a result of such a transformation is the ReceiveOutcomeBehaviour that can be viewed in Listing 62 in Annex 6).
    - Else, add in the action method of the behavior class the contents of the Activity attribute of the node (a result of such a transformation is the DecideResponseBehaviour that can be viewed in Listing 51 in Annex 6).

- Else, if the node's type is "AND" then it is transformed to a JADE *ParallelBehaviour* (it extends the jade.core.behaviours.ParallelBehaviour class). All the eligible sons of the node are added as threaded behaviours and the *ParallelBehaviour* ends when all its children have ended (a result of such a transformation is the _open_group_ManageMeetings_sequence_LearnUserHabits _close_group__forever__parallel_NegotiateMeetingDate_forever_Behaviour that can be viewed in Listing 46 in Annex 6).

- Else, if the node has two sons, the second of which has a transition to itself then the latter is the case of a behavior that will execute forever. Thus this node must be transformed to a behavior that will continuously instantiate its second son (the first is a node of type START, thus is ignored). This is achieved by transforming it to a *CyclicBehaviour* (it extends the jade.core.behaviours. CyclicBehaviour class) that checks if the eligible son has finished and if this is true it restarts it (a result of such a transformation is the NegotiateMeeting Date_forever_Behaviour that can be viewed in Listing 57 in Annex 6).

- Else, if the node has three sons, the second of which has a transition to itself then the latter is the case of a behavior that will execute one or more times. Thus this node must be transformed to a behavior that will continuously instantiate its second son (the first is a node of type START, thus is ignored) while a specific condition holds. This is achieved by transforming it to a *SimpleBehaviour* that checks if the eligible son has finished and then if the condition of the transition that has it as target is true it restarts it. If not the behavior terminates (a result of such a transformation is the _open_group_ DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group __one_or_more_times_Behaviour that can be viewed in Listing 43 in Annex 6).

- Else if the node has a son whose type is CONDITION then

    - If the node has four sons, then its third son is the case of a behavior that will execute zero or more times (see the template for the x* Gaia operator in Table 5). Thus this node must be transformed to a behavior that will conditionally instantiate its third son (the first is a node of type START, the second the one of type CONDITION). This is achieved by transforming it to a *SimpleBehaviour* that conditionally adds the sub-behaviour in its constructor and that checks (in its action method) if the eligible son has finished and then if the condition of the transition that has it as target is true it restarts it. If not the behavior terminates.

    - Else this node has a number of eligible sons one of which must be instantiated. It is transformed to a *SequentialBehaviour* (it extends the jade.core.behaviours.SequentialBehaviour class) and at its constructor it conditionally instantiates one of its sons (a result of such a transformation is the _open_group_ReadSchedule_or_RequestChange Meeting_or_RequestNewMeeting_close_group_Behaviour that can be viewed in Listing 49 in Annex 6).

- Else this node has a number of eligible sons that must be executed sequentially. This is achieved by transforming it to a *SequentialBehaviour* and adding all its eligible sons sub-behaviours (a result of such a transformation is the _open_group_DecideResponse_sequence_SendResults_sequence_Receive Outcome_close_group_Behaviour that can be viewed in Listing 44 in Annex 6).

All the automatically generated classes for the *PersonalAssistant* agent of the meetings management sample project are include in Annex 6 from Listing 42 to Listing 71. If the user has inserted the activity related to each node in java code format he has to denote this by starting the activity description with the string "/*Java code*/". In this case the code is inserted as-is in the action method of the SimpleBehaviour. Thus, code generation can come up to 100% of the needed code.

However, in normal projects it is expected that diverse technologies will be involved, in which case the programmer will have to edit the action methods of the simple behaviours. This was the case for the MARKET-MINER and ASK-IT projects where prolog code and web services invocations had to be integrated in the agent's code. The reader should note that MARKET-MINER was not implemented using JADE, however the same holds for any implementation platform.

Thus, the ASEME developer can generate all the needed classes for his project just by executing the "workflow.oaw" transformation workflow file in the Eclipse IDE. The resulting files are the JADE Agent and Behaviour descendant classes along with the variable holder classes, 31 files total for the personal assistant agent and they are shown in Figure 95.

In the same figure, the SendResultsBehaviour is depicted. It is worth discussing it along with its attributes and methods. The properties of the class are two holders for ACL messages and one holder for the Meeting class. These are initialized through its constructor. The action and done methods have been produced and an if-else statement has been introduced in the action method initializing the performative of the message to be sent. As the designer has not defined the conditions for sending one or the other message type the conditions in the if-else statement are filled with the comment "/*insert condition*/". Then the developer is reminded to insert more ACL message initialization code with a comment and the message is sent.

The final version of the "SendResultsBehaviour.java" is shown in Listing 18. The only editing that the developer needed to do to this file was to define the condition of the if-else statement (shown with grey background). This of course is a very simple message sending behaviour, however it is demonstrated that all the mundane code has been prepared for the developer.

Thus, among the 31 defined classes the developer will need just to define the action methods of 14 of them, seven of which are message send and receive methods, which will just require a final touch. Therefore, the developer will just have to write code for seven methods. Everything else has already been prepared by the ASEME tools.

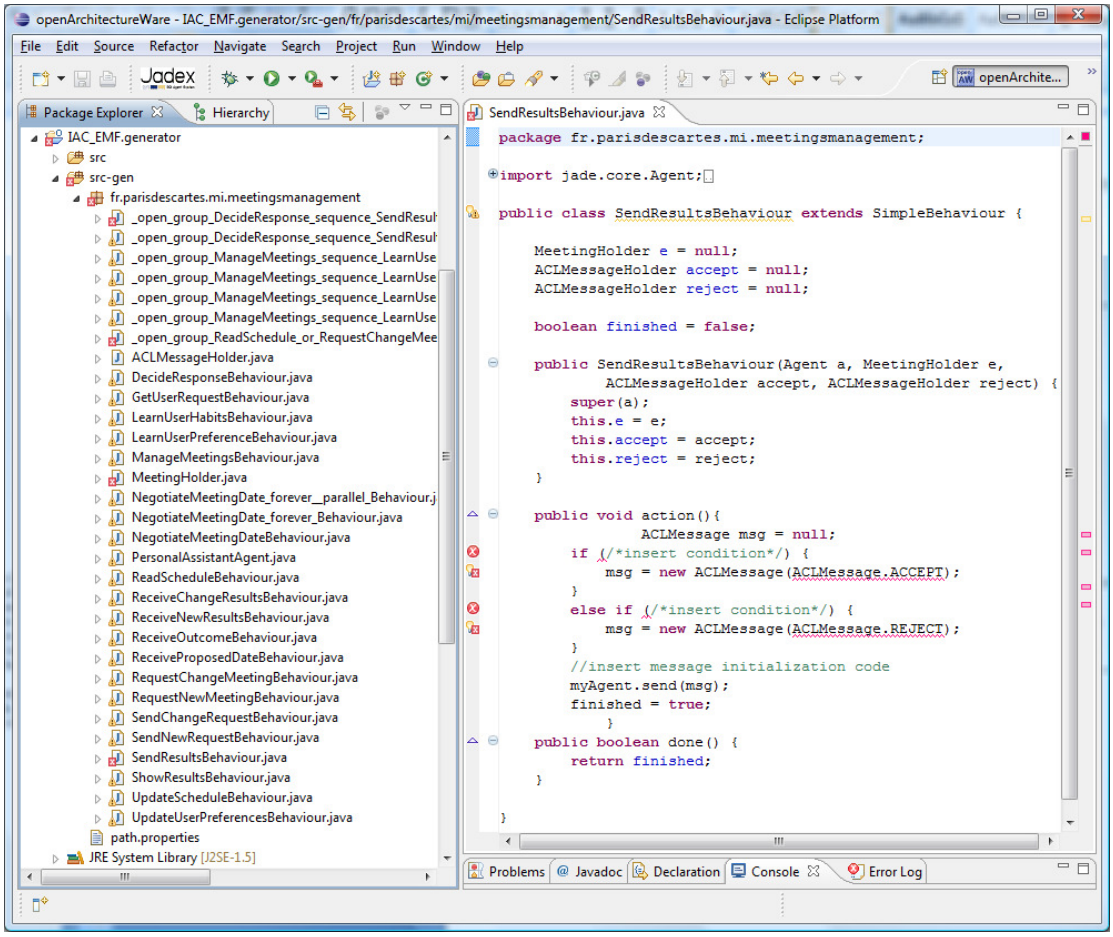


Figure 95. The automatically generated java classes for the personal assistant agent of the meetings management project.

Listing 18. The final file SendResultsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;

public class SendResultsBehaviour extends SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder accept = null;
  ACLMessageHolder reject = null;

  boolean finished = false;

  public SendResultsBehaviour(Agent a, MeetingHolder e,
      ACLMessageHolder accept, ACLMessageHolder reject) {
    super(a);
    this.e = e;
    this.accept = accept;
    this.reject = reject;
  }

  public void action(){
            ACLMessage msg = null;
    if (accept.getACLMessage()!=null) {
      msg = accept.getACLMessage();
```

```
    }
    else {
      msg = reject.getACLMessage();
    }
    myAgent.send(msg);
    finished = true;
          }
  public boolean done() {
    return finished;
  }

}
```

Finally, it is worth showing how a capability has been developed as a software module. In Listing 19 the automatically generated "NegotiateMeetingDateBehaviour .java" file is presented. It is the implemented personal assistant's part of the "Negotiate Meeting Date" protocol. The reader can see that it defines the ACL message holders for the types of messages that it handles and which it then uses for adding its children behaviours to the agent scheduler. This behaviour along with its children behaviours (i.e. the _open_group_DecideResponse_sequence_SendResults_ sequence_ReceiveOutcome_close_group__one_or_more_times_Behaviour, _open_group_ DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group_ Behaviour, DecideResponseBehaviour, ReceiveProposedDateBehaviour, ReceiveOutcome Behaviour, SendResultsBehaviour, UpdateScheduleBehaviour) and the used variables (MeetingHolder and ACLMessageHolder) can be used by any future JADE agent that wants to participate as a personal assistant to the "Negotiate Meeting Date" protocol. He just has to import the identified classes of the *fr.parisdescartes.mi.meetingsmanagement* Java package and add the NegotiateMeetingDateBehaviour to his agent's behaviour scheduler.

Listing 19. The generated file NegotiateMeetingDateBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class NegotiateMeetingDateBehaviour extends SequentialBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder accept = new ACLMessageHolder(this);
  ACLMessageHolder inform = new ACLMessageHolder(this);
  ACLMessageHolder propose = new ACLMessageHolder(this);
  ACLMessageHolder reject = new ACLMessageHolder(this);

  public NegotiateMeetingDateBehaviour(Agent a, MeetingHolder e) {
    super(a);
    this.e = e;

    addSubBehaviour(new ReceiveProposedDateBehaviour(this.myAgent, e,
        propose));
    addSubBehaviour(new
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group__o
ne_or_more_times_Behaviour(
        this.myAgent, e, accept, inform, propose, reject));
    addSubBehaviour(new UpdateScheduleBehaviour(this.myAgent, e, inform));

  }
}
```

5.3 The ASEME MDE Process

Having defined the models and transformations participating in the ASEME MDE agent development process it is now possible to present a diagram showing this process. It is a more compact process than the one shown in Chapter 4 as it only includes the models of the agent abstraction level of ASEME. Thus, the SPEM diagram shown in Figure 96 summarizes the process specified in this chapter for agents' development.

The activities for editing a model, e.g. "Edit the SAG model" are those performed by the human developer, while the transformation activities, e.g. the "SAG2SUC" are automated. The modeler uses one model until the "SRM2IAC" activity where one IAC model is created for each agent type. Then the modeler must work in each IAC model separately. Finally, in the "IAC2JADE" activity one agent class and many behavior classes are automatically generated for each IAC model instance.

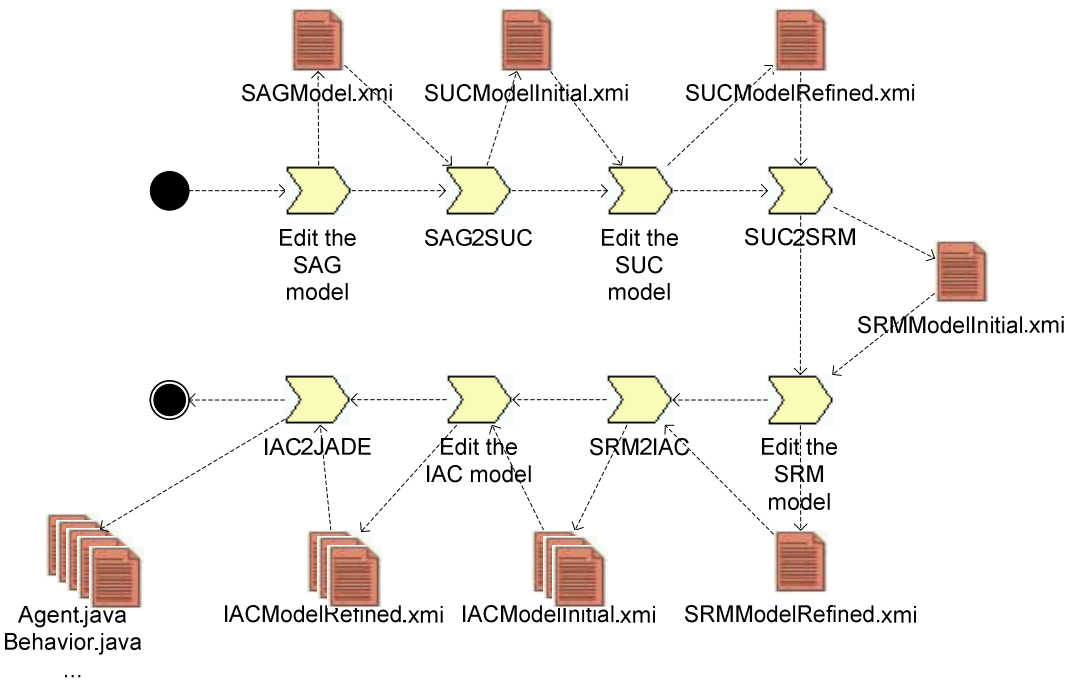


Figure 96. The ASEME MDE Process for Agent Development.

Chapter 6

Process Modeling

There exist situations in massive multi-agent systems where the number of executing agents is very important for measuring future system stability and performance. The main contribution of the paper of Rana and Stout (2000) is to highlight the importance of combining performance engineering with agent oriented design methodologies, to design and build large agent based applications. In such an application, for example, in the e-Marketplace middleware framework of Yamamoto and Nakamura (1999), the performance decreases as the number of shop agents (a type of agent that they use) increases. This means that if a site hosts many shops and, thus, agents, the site needs a more powerful computer. These situations can be simulated and optimized using a process transformation approach.

This chapter is about defining a transformation process of the AMOLA design models (statecharts) to process models. These can be used at the verification and optimization phases of ASEME and they can be present in all development iterations since the AMOLA models of the design phase are statecharts. Statecharts can be transformed to process models, since all states represent an activity that is executed by a specific agent role resource. Both statecharts and process models are supported by commercial and open source tools (e.g. STATEMATE for statecharts and Micro Saint Sharp for process models) that allow for simulation and thus, can greatly aid the verification and optimization phases. Micro Saint Sharp can also be used for optimizing process models.

6.1 Transforming IAC and EAC Models to Process Models

Both in the societal level of abstraction and the agent level of abstraction in the design phase of the ASEME methodology, the statecharts of the inter-agent protocol and intra-agent control models can be transformed to processes. Using the formalism proposed by Lonchamp (1993), for the intra-agent control model we find the concept of *Activity* similar to the one we have defined. We assign to it, as a resource, the software library that realizes the underlying functionality. Similarly, capabilities are *Tasks* whose resources are the relevant modules. Finally, for the inter-agent protocols model the resources for the protocols states activities are the respective agent types. In this way the ASEME design models can be simulated and optimized using relevant tools like SIMPROCESS (April et al., 2006) or Micro Saint Sharp (Bloechle and Schunk, 2003).

To illustrate the process of transforming a statechart to a process model one can use, for instance, the open source Intalio tool (see http://bpms.intalio.com/ for more details). The statechart in Figure 71 is transformed to the business process diagram presented in Figure 97. In Intalio, the message reception activities are represented as circles with an envelope inside. The circle with the square that starts the *Meetings Manager* process resembles the need for some conditions to be satisfied so that the process starts. Finally, a diamond with an X resembles an exclusive choice (either one or the other). The states of the inter-agent control model (see Figure 71) are transformed to processes in the business process diagram. The transitions between states are transferred as they are except in the case of transitions that are enabled by an inter-agent message event. The latter are becoming transitions from the sending process to the receiving process. The conditions on the transitions can be represented in Intalio and the process can be directly simulated, or even deployed to executable code (only if the activities can be implemented with the available Intalio tools). The *meetings manager* role interacts with all the *personal assistant* roles in the way presented in Figure 97.

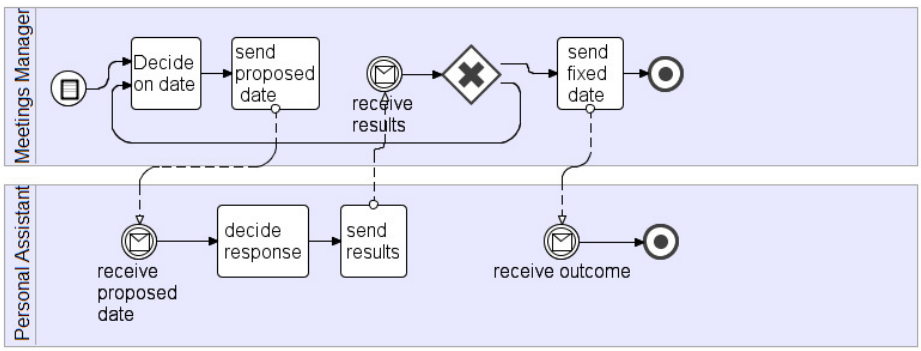


Figure 97. The Negotiate Meeting Protocol Process

6.2 A Case Study: The ASK-IT project

This ASK-IT project's agent system was developed using the Gaia2JADE process for multi-agent systems development, the predecessor of ASEME. Thus, the output of the analysis phase was the Gaia roles model. Initially, there were two reasons for simulating the ASK-IT system. The first was that the ASK-IT service providers needed to know if the system can satisfy non-functional user requirements, one of which was *the delivery of the service within ten seconds*. The frequency of service requests was calculated to be one request per 30 seconds. The second was to find out how would the system scale when service demand increased for use in preparing the project's exploitation plan.

The intra-agent control model was transformed to a process model using the approach described in the previous paragraph. The tool that was used was the Micro Saint Sharp Gold edition. Micro Saint Sharp is a flexible discrete-event simulation software package for modeling all types of processes (http://www.maad.com/index.pl/micro_saint). It allows for modeling a process where activities can be complex, i.e. be analyzed to more specific ones in various abstraction levels. In this way it is possible to transform a non BASIC IAC node to a complex activity (or task) and BASIC IAC nodes to simple tasks.

It is out of the scope of this thesis to present the way that the Micro Saint Sharp tool works, however the reader will understand its basic functionality while reading through this section. The first part of the Micro Saint modeling process is to identify the process involved and set some goals.

Then the process elements must be identified in the form of simple and composite tasks. In the highest level of abstraction there is the multi-agent system. The Personal Assistant (PA), Broker (BR) and the Added-Value Service Provider (AVSP) agents participate in the simulated scenario. The goal is to analyze the process followed during the execution of the Request for Services protocol on the server side (time to respond to the PA). The high level view of the process is shown in Figure 98. The *start* task resembles the user that initiates a service request. Arrows in the figure show the direction of the process execution.

Zooming in the PA the process is very simple as the only part of the PA relevant to this study is the Request for Services protocol initiator part. The protocol is initiated when the PA sends a request message and is terminated when he receives the response. Figure 99 shows this part of the process. The higher level processes, with which the low level processes are connected, are shown as squeezed hexagons with a dotted line border.

The broker (BR) is a more complex process and resembles the broker agent's IAC (see Figure 53). The BR is shown in Figure 100 having put all the simple processes in the same level for a more effective presentation.

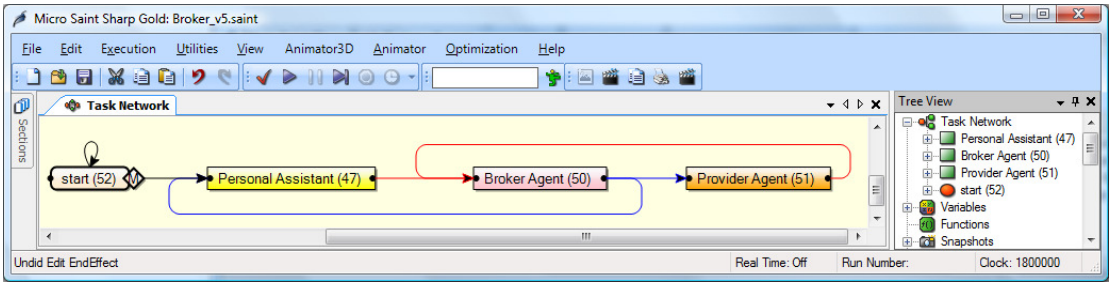


Figure 98. The ASK-IT Request for Services Protocol participant agents in a high level process view in Micro Saint Sharp.

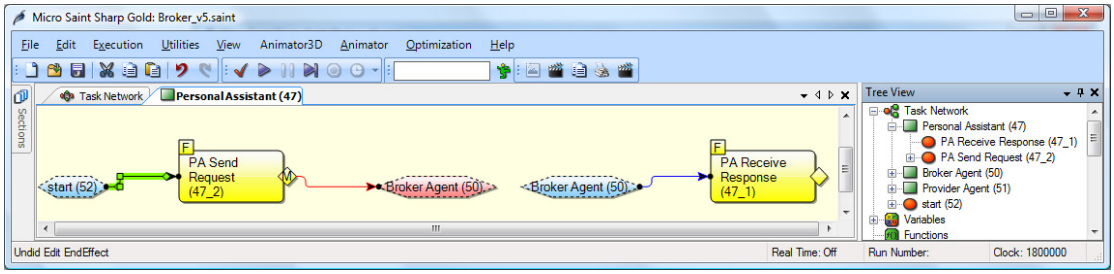


Figure 99. The Personal Assistant internal process.

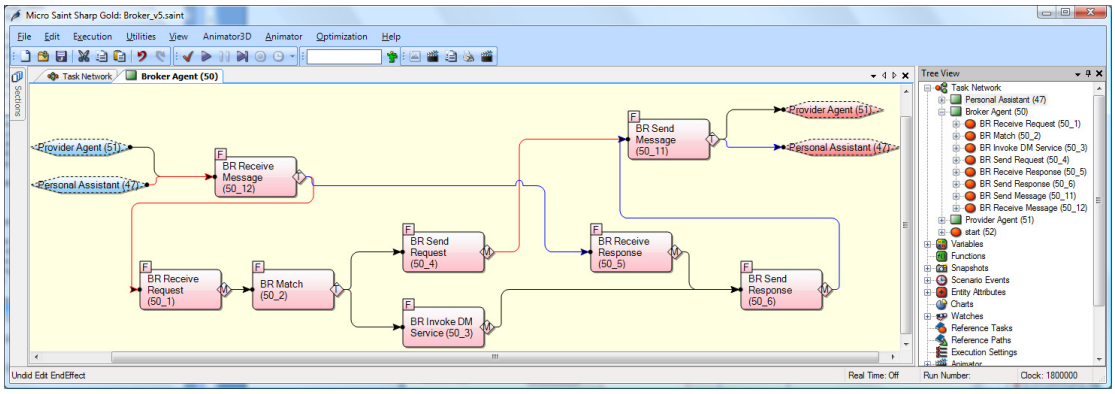


Figure 100. The Broker agent internal process.

Finally, the Added-Value Service Provider (AVSP) is presented in Figure 101. The process model resembles the AVSP's IAC model (the reader can see the AVSP's liveness formula in Figure 48). When the AVSP receives a message, it is either a mapping or a routing request. In order to satisfy this request the AVSP uses several simple services offered by the broker. Therefore, the reader can see that the AVSP

not only sends to the broker the service response but also service requests. The AVSP offers an added value service by orchestrating many simple service requests on behalf of the PA and returning only the most relevant results to the serviced user profile.

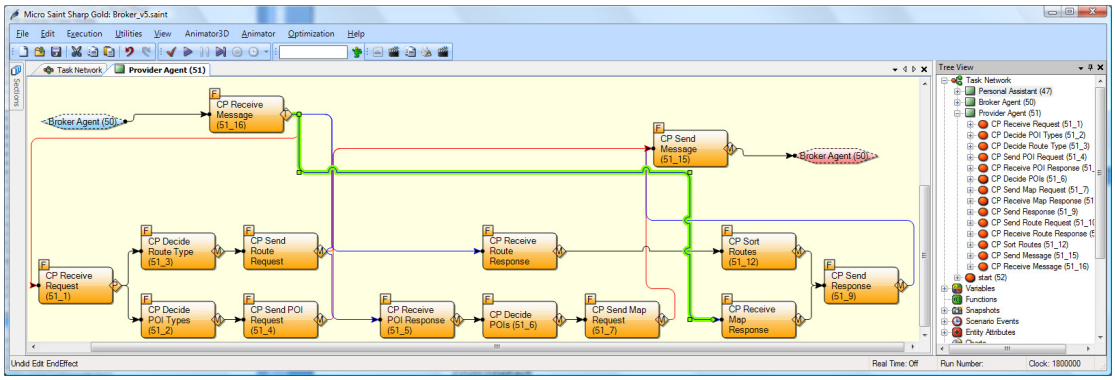


Figure 101. The complex provider agent internal process

Before presenting the Micro Saint simulation process the reader needs to become familiar with some concepts. Messages in Micro Saint are simulated as *entities* that traverse the system's tasks. A task can take time to handle an entity, can produce more entities and can act upon the properties of an entity. Whenever an entity is about to enter a task a release condition is tested. If the release condition is true the task can start execution. Then the modeler can define beginning and ending effects for the process. In Figure 102 these data are defined for the BR receive message process. Thus, this task can only start if there is an available CPU, which it then uses as a resource by consuming it while executing (the beginning effect lowers the number of CPUs by one, while the ending effect raises this number by one).

The reader should note that this task has not been defined in the Broker's IAC model (see Figure 53). It was introduced so that the Agent platform's message receiving activity could be simulated (and also measured for the time it takes to execute). After executing a message receiving activity many times in the JADE platform it was determined that the time of its execution would be modeled with a gamma probability distribution function with a mean value of 2 milliseconds and a standard deviation of 2 milliseconds. This information is inserted in the next tab of the Micro Saint task definition dialog presented in Figure 103. The gamma distribution is often used to model nonnegative random variables and the Micro Saint Sharp User Guide (Alion Science and Technology, 2008) suggests using it for task times when a task cannot be done much faster than the mean time, yet could take much longer.

The third important thing to define for a task is the path that the entity will follow after leaving it. For the BR receive message task there are two possibilities, one is to go to the BR receive request task and the other is to go to the BR receive response task. The method for selecting where to branch is to check two important properties of the entity, the sender and the performative. Thus, if the sender is the PA then it is

a new request. It is the same if the sender is the AVSP and the performative is “Request”. In all the other cases (i.e. the sender to be the AVSP and the performative to be different to “Request”) it is a service response. This information is inserted to the *Paths* tab of the Task dialog as shown in Figure 104. The reader should note that the language used for writing the expressions in the dialogs is the C#.

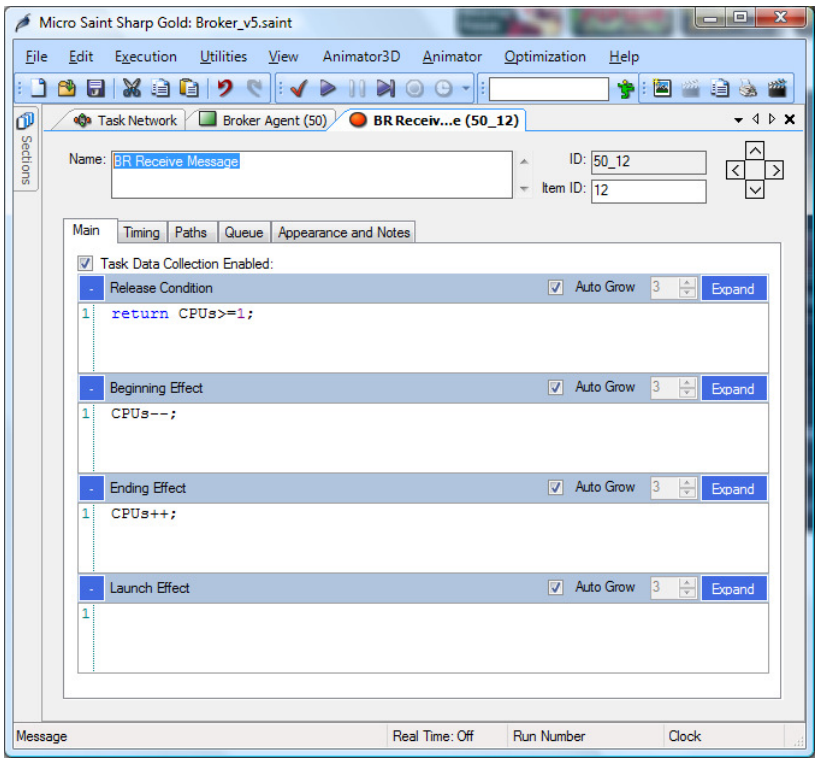


Figure 102. The main properties of the BR receive message task.

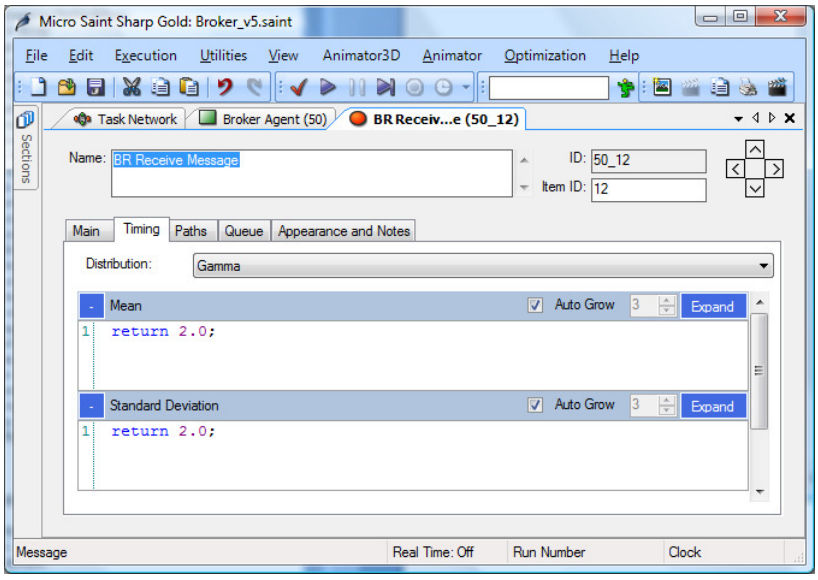


Figure 103. The timing of the BR receive message task.

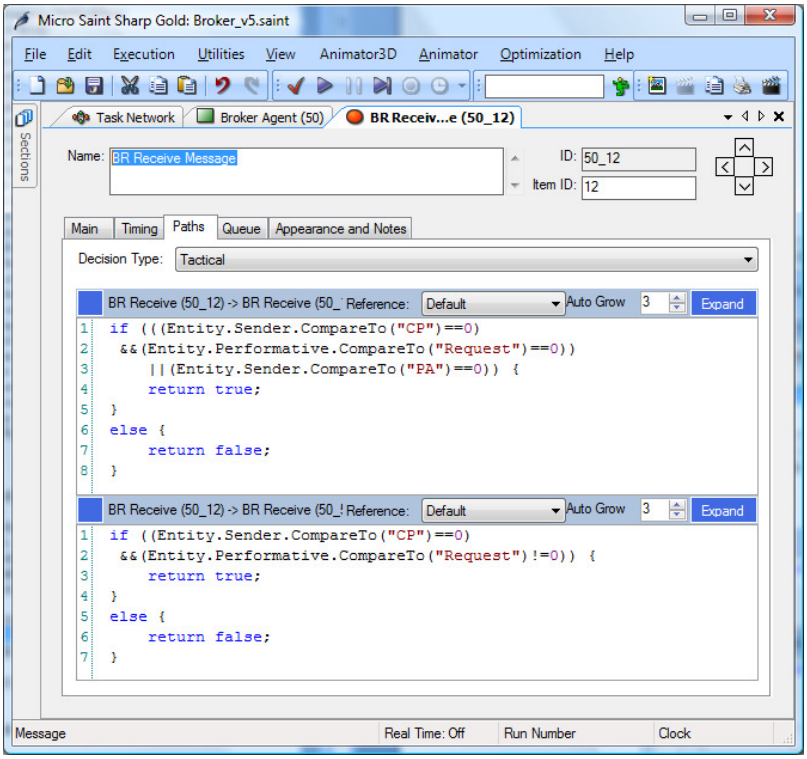


Figure 104. The execution paths after the BR receive message task.

All the information needed for reproducing this simulation is provided in Annex 8. After all the tasks have been completed, the simulation is ready to execute. Initially, the model had the restriction posed by the IAC model that the broker could have at most 10 instances of the request for services protocol executing. In field tests everything had worked fine. In the simulation, though, when more than one request per 10 seconds was issued a side effect was recognized. If 10 requests were sent within a short time (i.e. before the broker had completed any one of them) then the broker had consumed his resources and when the AVSPs sent their requests he could not service them and in this way the system failed to respond. Thus, after the simulation it was decided to allow for unlimited number of instances of the request for services protocol for the broker to execute in parallel.

After fixing this issue, the simulations were done again and the results for a mean request arrival time of 30, 15 and 5 seconds are presented in Figure 105. Each of the three cases is presented in table form with three columns:

- *Clock*: The clock ticked in milliseconds and the overall simulation time was 30 minutes (or 1,800,000 milliseconds). The table data are ordered according to the time passed.

- *Entity.Tag*: Each entity gets a tag when it is born starting from one. As the reader can see the entities do not complete their lifecycle always in the order with which they are introduced into the system.

Clock	Entity.Tag	TimeSystem
49026,15	1	7277,12
113160,65	2	5395,60
122675,52	3	3390,57
130159,90	4	5189,93
163688,24	5	4794,08
171308,33	6	3112,60
198684,29	7	7680,04
238609,75	8	7727,28
245741,33	9	7134,24
270145,82	10	4853,68
341097,11	11	4980,08
416899,82	12	8392,50
460855,57	13	8381,43
470306,39	14	3384,41
484243,87	15	4799,52
495758,04	16	5270,09
503396,48	17	3575,99
529562,55	18	6983,56
562412,88	19	7925,93
564501,02	20	3156,55
618883,21	21	3550,63
645772,68	22	8274,06
715621,55	23	1436,23
731482,07	24	6605,36
732096,97	25	4908,03
808946,44	26	3922,34
826972,03	27	7062,80
832303,96	28	3847,17
874068,55	29	4595,74
880236,84	30	3437,46
929295,34	31	8090,11
984276,00	32	7462,90
993094,23	33	5377,21
1018441,90	34	7850,43
1073659,41	35	1887,93
1080287,78	36	6927,05
1122869,95	37	2976,49
1153465,57	39	2183,59
1154694,79	38	6527,49
1192207,11	40	4885,26
1219386,82	41	7632,16
1222371,41	42	5921,83
1224564,19	43	5369,14
1249000,68	44	1882,38
1275700,67	45	3551,47
1303171,22	46	8212,72
1305210,03	47	3737,15
1307980,75	48	2493,82
1355722,05	49	1847,44
1392124,14	50	6002,85
1429074,61	51	4035,66
1440766,13	52	5471,22
1491243,48	53	5731,98
1505447,66	54	2445,92
1533004,83	55	4740,00
1613091,27	56	7460,34
1634274,43	57	6953,50
1655310,52	58	6720,44
1669748,92	59	3810,26
1689609,06	60	7560,49
1744526,56	61	3451,17
1753827,64	62	7578,12
	mean	**5287,48**
	max	**8392,50**
	min	**1436,23**

(1)

Clock	Entity.Tag	TimeSystem
28151,63	1	7277,12
59278,13	2	5395,60
63057,15	3	3414,68
66809,83	4	4324,84
77612,41	5	6166,61
78981,77	6	2959,68
84855,49	7	4009,88
92089,79	8	8600,19
109838,12	9	5250,49
115235,81	10	5706,90
116749,71	11	3744,22
141939,06	12	3882,45
148529,24	13	8584,19
166133,68	14	8887,41
170650,91	15	4829,10
181132,72	16	3688,30
190507,40	17	9176,40
216021,45	18	6700,01
223808,66	19	8951,57
231335,16	20	6367,71
233961,96	21	2492,14
256653,05	23	3659,99
260004,56	22	8446,97
263131,54	24	2917,45
303458,48	25	3457,65
339815,88	26	4931,41
361188,47	27	3977,80
377896,45	28	4239,13
399984,56	29	3038,50
418043,70	30	8424,35
452162,35	31	2001,86
…	…	…
1250101,38	80	3774,36
1257236,39	81	1736,37
1263753,81	82	6488,45
1294529,23	83	6961,88
1324085,38	84	3601,68
1325147,33	85	4422,76
1330766,44	86	6703,55
1362124,09	87	2686,61
1371852,04	88	5640,76
1380704,30	89	7537,03
1406984,27	90	3170,21
1407864,18	91	2967,99
1449966,67	92	4942,72
1456403,68	93	6094,67
1489325,30	94	2963,67
1502884,51	95	4201,85
1529002,40	96	3591,10
1541228,19	98	7370,96
1541423,66	97	8208,82
1627889,10	99	4670,53
1632800,18	100	2418,23
1645122,77	101	7013,93
1671344,04	102	8328,92
1672775,36	103	6743,40
1686752,98	104	892,86
1699000,65	105	5861,38
1734153,62	106	2524,53
1741522,45	107	7140,02
1765053,74	108	5121,95
1773298,50	109	3077,34
	mean	**5169,84**
	max	**12576,34**
	min	**892,86**

(2)

Clock	Entity.Tag	TimeSystem
14235,29	1	7277,12
22357,16	3	2476,33
25631,04	2	7670,19
30982,75	4	6568,39
53378,06	5	11382,45
53830,09	7	7382,00
54656,10	6	9264,55
61487,94	8	5886,32
68377,42	10	4261,89
70890,55	9	8868,01
74081,17	11	6620,95
81922,93	12	8029,23
84704,60	13	2492,73
91738,01	14	5151,96
95240,72	15	7634,97
104651,09	16	5809,45
108676,87	17	4636,24
115433,39	19	6975,63
116947,57	21	7231,67
117145,06	18	9708,36
119441,36	20	10271,44
124036,87	22	4792,37
135476,98	23	5443,55
136645,40	24	3230,17
140860,27	26	3283,42
144217,70	25	10199,27
148708,67	27	4874,29
159586,09	28	5737,36
180404,07	29	2364,08
184408,86	30	1791,41
196272,44	31	8041,10
…	…	…
1689001,86	304	12041,25
1689051,68	305	6633,93
1691510,82	307	5228,11
1695248,79	306	9607,74
1695369,61	308	2095,65
1704413,24	309	10056,95
1705672,76	310	10288,13
1708012,14	311	11679,76
1711027,00	312	5012,96
1713290,56	313	6311,91
1713517,16	314	4269,67
1719886,36	316	4635,73
1723296,11	317	3917,51
1723404,88	315	8294,86
1726715,10	318	4023,59
1730569,43	319	4783,14
1736649,54	320	5316,99
1745232,75	321	7182,12
1753182,26	322	14713,14
1754918,09	325	9784,50
1754921,23	326	4056,10
1754953,04	323	11909,92
1756532,40	324	12658,17
1761889,24	327	1900,82
1769177,81	328	4942,09
1772898,38	329	3567,35
1780128,14	330	5123,63
1785146,16	331	3948,79
1796496,70	333	8623,13
1796550,38	332	8886,97
	mean	**6669,70**
	max	**14713,14**
	min	**1791,41**

(3)

Figure 105. The Broker agent's response times (mean, maximum and minimum service values) when a message is coming in average every 30 seconds (1), 15 seconds (2) and five seconds (3).

- *TimeSystem*: The total time that an entity spent into the system. This time is calculated from the moment that the message leaves the PA Send Request task (see its ending effect in Annex 8) until the moment that it is received by the PA Receive Response task.

At the bottom of the columns three figures are calculated, the mean, the maximum (max) and the minimum (min) *TimeSystem* value for all entities. The simulation proves that the requested non-functional requirements are met for the 30 seconds request arrival time even for the worst case scenario. The 15 second and 5 second columns in Figure 105 show how the system scales for increasing demand.

Thus, using a process model and simulation during the verification and optimization phases the modeler can:

- Determine if the system meets its requirements
- Determine how the system would scale
- Identify errors in system conception and propose strategies for resolving them either through a next development iteration (including risks calculation and different technology use) or by directly returning to the phase that introduced the error and restarting from there (useful in agile development). In this case the modeler returned to the liveness formula of the broker and replaced the $|\text{request for services } SP^{\omega}|^{10}$ element with $|\text{request for services } SP^{\omega}|^{MAX\_INT}$
- Optimize the system using tools in the market regarding resource allocation and consumption (in this case the only resource was the CPU). An optimization model in OptQuest for Micro Saint Sharp has three major elements:
    - Decision variables. Decision variables are quantities over which you have control, such as the amount of product to make, the number of dollars to allocate among different investments, or which operational rules to select from a limited set.
    - Constraints. Constraints describe relationships among decision variables that restrict the values of the decision variables. For example, a constraint might ensure that the total amount of money allocated among various investments cannot exceed a specified amount, or it might ensure that no more than one operational rule from a certain group can be selected.
    - Objective. The objective presents a mathematical representation of the optimization model's objective, such as maximizing profit or minimizing cost, in terms of the decision variables.

Chapter 7

Future Perspectives

This first ASEME methodology specification sets the foundation for the establishment of ASEME as an AOSE methodology but can also be considered as a start of a multi-path research effort.

7.1 Further Evaluate and Expand ASEME

A first evaluation of the methodology has been done through the MARKET-MINER project, however, more results can be obtained that can help ASEME establish itself as a reliable AOSE methodology.

An important aspect of the ASEME establishment is to implement an eclipse add-on that can guide the developer through the development steps and supply him with nice graphical editing tools. The *Sample Reflective Ecore Model* editor or the *epsilon exceed* editor of Eclipse that were used in this thesis provide a minimal functionality. The modeler should be able to see and edit his diagrams using the right graphical representation like in the case of the Rhapsody tool.

Another possibility is to integrate to the code generation possibilities the ability to automatically generate code for interfacing with standardized environment-related systems such as an OSGi service, a web service and human-machine interfaces.

The author's work in integrating an OSGi service oriented framework with the JADE environment can be extended in order to develop Xpand template files for generating the additional components and agents needed for this task. Challenges

include the automatic definition of the services descriptions based on an ontology (which could be externally defined) and the deployment in different platforms (e.g. PDA) that have different capabilities.

Integration with web services is another aspect and the JADE framework already has a relevant package which can be used, again with the challenge of automatically integrating a foreign ontology.

The need for a human-machine interface can automatically be derived by the SUC model role types. Then, the different transitions exiting a user interface use case can be transformed to different choices for the user in an automatically generated form.

Finally, the ASEME implementation phase can be enriched with transformations to other agent platforms such as JACK, which is used by many existing AOSE methodologies. The following are the main concepts modeled by JACK (Winikoff, 2005):

- *Agent*: The agent defines the events that he can send or receive, the data (like beliefsets) he handles, and the plans and capabilities he has access to.
- *Beliefset*: A beliefset is a relational database storing variables (beliefs). The change of the value of a belief can generate an event which can trigger an agent plan execution.
- *Event*: An event is an occurrence in time that represents some sort of change that requires a response. Events model incoming inter-agent messages, new goals being adopted, and, generally, information received from the environment.
- *Plan*: A plan is a "recipe" for dealing with an event type. Plans indicate which event they handle, a context condition which describes in which situations the plan can be used, and a plan body, which is what is actually executed.
- *Capability*: A capability is a modularization construct. It allows to group plans and beliefsets. It is very similar to the agent class. One could say that the agent is the top level capability.

The challenge of transforming the IAC to the JACK platform is considered not to be trivial, however a few guidelines can easily be extracted. The JACK plan can be seen as a BASIC IAC state including the event and condition for the transition that will have it as target. The state activity is the plans body. Capabilities define the capability level variables (as beliefs) and define which plan is executed when (like the OR states). There can be parallel plan execution (AND states) defined by plans. Variables participating in IAC transition expressions (the AMOLA intra-agent messages) will be included in the JACK agent beliefset and occurring changes to be defined as events triggering the interested plan.

Another interesting possibility is to define the M2M transformations for the Gaia, Tropos and UML models so that existing system models can be ported to the AMOLA language and further elaborated using the ASEME process. This way, the process of

integrating a method fragment (of Gaia, Tropos or UML) to ASEME will be automated.

Finally, the most common agent interaction protocols (e.g. those defined by FIPA) can be defined as EAC models allowing the developers to use them automatically, as the PASSI methodology does (Cossentino, 2005).

7.2 Research Directions

7.2.1 Automate Software Development

One of the most important challenges is to define knowledge for enriching the models at each development phase. In the SUC model the knowledge should be related to how to decompose a general task to specific ones. Moreover, the specific tasks must be associated with a technology for achieving them. In the SRM model the knowledge must be related to how to assemble the previously defined tasks to realize a capability.

In the IAC model the knowledge must be related to variables definition and documenting the functionality in a format as close as possible to the developed code. As the IAC is platform independent it is difficult to point out a specific language for describing the functionality, however a pseudocode or any platform dependent implementation could be used.

7.2.2 Self-Assessment and Self-Healing Agent Capability

The IAC model can be used by a new module of the agent which can keep track of the occurring transitions and detect anomalous or not frequent situations. For example, the ASK-IT broker agent can keep track of the web service invocation results and suddenly realizes that whenever it invokes a web service he always gets a failure result, while normally he get a failure in 3% of invocations. This could mean that its web service invocation component has failed, or it is outdated and needs an update. This meta-information on the agents executing lifecycle can be very useful if it can be automated in the agent's code generation.

7.2.3 Automate the Process Model Generation

An important research direction is towards automating the IAC model transformation to a process model. The problem with this issue is that unlike the software development community were metamodels for various model types are generally standardized, process models are not widespread and thus, there are no

tools available in the market supporting a common format. However, a more focused research effort in this direction might bring about something like a process metamodel for the eclipse environment along with tools for simulation similar to those in the market.

7.2.4 Incorporate Organizational Rules

Having defined protocols and allowed agents to incorporate them in their capabilities the next challenge is to regulate the ability of an agent to participate in a protocol. Several authors in the literature have worked on organization rules and norms for regulating agent-based interactions. The question is whether this is a design issue or a runtime issue.

Zambonelli et al (2003) propose that organizational rules should be defined starting from the analysis phase of a system. These rules can define when an agent is allowed to participate in a protocol with a given role. Moreover, these rules are defined in an organizational level, not in the agents' definitions. In ASEME they could be incorporated in the EAC model in the form of constraints in the first transition of each participant. However, in this case there must be a way for these rules to find their way to implementation independently from the produced agent code but constraining their execution. Sowmya and Ramesh (1998) define a logical specification language capable to express safety and liveness properties for statecharts. It would be interesting to see how to define such rules in the EAC model and also transform them to a platform implementation (such as JADE) with the capability of preventing agents from starting a protocol or participating to it.

Alberti et al. (2006) propose a tool executing on an agent platform that is logic based and check the agents for compliance with the interaction protocols. Researchers have proposed temporal logics for verifying statecharts in the past and one of these could be adapted to verify the EAC model instances on the runtime. Moreover, the work of Alberti et al. (2006) could be extended for also checking the authorization of an agent to initiate or participate to a given protocol.

Chapter 8

Conclusion

Summarizing, this thesis presents the Agent Systems Engineering Methodology (ASEME) and the Agent Modeling Language (AMOLA). AMOLA defines the System Actors Diagram (SAG) and the Requirements Per Goal (RPG) models for the ASEME requirements analysis phase, where the modeler defines the actors and their goals, the latter associated to requirements, using the Eclipse tool.

Then, AMOLA defines the System Use Case (SUC), Agent Interaction Protocol (AIP), System Roles Model (SRM) and Functionality Table (FT) models at the Analysis phase. The modeler uses the SAG2SUC transformation to automatically instantiate a SUC model from the previous phase SAG model (transforming actors to roles and goals to use cases). Subsequently, he uses the Eclipse tool in order to decompose the general use cases to specific ones using the “include” UML relationship.

Continuing the analysis phase, the modeler uses the SUC2SRM transformation to automatically transform the SUC model to a SRM (transforming general use cases to capabilities and specific use cases to activities). Then, he associates the activities to functionalities and uses the Eclipse tool to write liveness formulas.

In the ASEME design phase AMOLA defines the Inter- and Intra-Agent Control models (EAC and IAC respectively). The SRM2IAC transformation automatically transforms the liveness formulas to statecharts. The modeler needs to use the Eclipse tool to define the transition expressions and variables related to each node.

In the ASEME implementation phase the modeler chooses a platform and instantiates the design phase models. Here-in the automatic transformation of the IAC model to agent program for the JADE platform has been demonstrated. The next step for ASEME is the development of a graphical tool that will aid the modeler to

navigate through the different development stages making transparent the use of the projects defined in Chapter 5.

At this point it is useful to also summarize the issues related to the novelty of this thesis and its contribution to the state of the art. First of all, AMOLA defines the intra-agent control (IAC), whose novelty is to allow the modeling of interactions between the different capabilities of an agent. Moreover, through the possibility to define alternative modes of such interactions, it allows the implementation of different behaviors (or profiles) for an agent. For this purpose, the statecharts and their orthogonality feature are used in an original way.

Another result of this thesis is the definition of the grammar for constructing Gaia liveness formulas extended with a new operator allowing for specifying a number of parallel instances of the same behavior. This result can be used by any practitioner of the Gaia methodology, if he wants to define a transformation from the liveness model to any kind of model (like in this thesis where it is transformed to a statechart).

AMOLA has the unique ability to model both inter-agent protocols (EAC) and agent capabilities (IAC) using the same formalism, i.e. the statecharts. This is one of the most important aspects of this work. Thus, integration of capabilities and protocols is seamless and so the individual and social aspects of an agent are represented in a natural unified way. This is why all participants in protocols are defined as different orthogonal components.

The IAC model provides an architecture that accommodates the extension of conversation policies that Moore defined in a theoretical way. According to this approach the agent uses the repository of conversation policies (or protocols that he implements) as orthogonal components sharing specific variables. Thus, while a protocol executes another can be launched in parallel and influence the outcome of the first by changing some of its variables.

ASEME defines an original recursive algorithm for transforming Gaia liveness formulas to statecharts. This result can be used by Gaia methodology practitioners “as is”.

The output of the design phase (IAC) is platform independent as it adheres to the well known language of statecharts for which numerous CASE tools are available in the market. The use of one of them, i.e. Rhapsody has been demonstrated.

The models of AMOLA can lead to agent development without imposing constraints on how the mental model of the agent will be represented (in contrast to most existing methodologies, like, e.g., Ingenias and Prometheus). The implementation of a BDI architecture using the IAC model has been demonstrated.

ASEME defines three clear abstraction levels throughout the development phases, a unique originality in contrast to all other methodologies.

ASEME defines a different meta-model for each development phase while all other methodologies have a unique meta-model covering all the software phases. This

metamodel is more complex and difficult to understand what information is added at each phase. 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.

ASEME is the first AOSE methodology to include three types (M2M, M2T and T2M) of transformations.

AMOLA clearly defines and uses the abstractions of agent, capability and functionality for aiding the system analysis phase. ASEME models agents out of real world actors and not like in most methodologies (usually those that end by proposing a way to produce JACK code, like Tropos and Prometheus) where agents are simply grouped capabilities with arbitrary criteria, or in MaSE where system goals are developed as agents.

The transformation of the IAC or EAC to a process model allows for scaling the results and observing the system in different levels of abstraction through simulation.

ASEME can be considered as an agile process allowing for rapid prototyping needing in the fastest case (the one presented herein) the editing of just four models (SAG, SUC, SRM, IAC). Hirsch (2002) proposes an agile version of the Rational Unified Process (RUP) where of the 80+ artifacts of RUP, only 10 to 12 are used on small projects. Agile PASSI uses seven models.

Each ASEME capability can be implemented as a different software module. Repositories of agent interaction protocols implementations can be defined and integrated in new agent designs as capabilities.

The inter-agent control (EAC) model defines both agent activities and exchanged messages in an agent protocol. In other methodologies, a protocol definition includes a number of defined messages and an order of sending and receiving them, not the functionality achieved by each participant within the protocol. ASEME provides a complete definition of the participation of a role in a protocol which can be reused and integrated with an agent's capabilities as sub-state of an OR-state or AND-state in the IAC model. Then, the modeler can choose to use any technology he wants for realizing the protocol activities.

ASEME uses non-functional requirements to influence the type of architectural solution and technologies that must be applied (see Garcia et al., 2006). Reading the functionality table a project manager can identify the competencies needed for his implementation team. Moreover, he can argue for or against the use of a specific technology also identifying the risks related to his choices (e.g. one technological solution may be more robust but also more expensive). The only methodology that allows for modeling non-functional requirements is Tropos, which defines soft goals for capturing them. However, Tropos does not show how these influence the development process afterwards. In the point of view of this work non-functional requirements do not add tasks and plans to a system but influence the way that the

functional requirements will be met (selection of technologies, needed development effort, and cost of equipment).

ASEME has already been used for modeling a real world agent system (see Spanoudakis and Moraitis, 2008) with a successful evaluation.

Numerous possibilities exist for extending this work and it is one of the intentions of the writer to try to inspire other researchers in delving into one of the identified paths.

Annex 1.

References

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. (2006). Compliance verification of agent interaction: a logic-based tool. Applied Artificial Intelligence, 20(2-4), 133–157.

Alion Science and Technology, Micro Saint Sharp User Guide, 2008, http://www.computing.surrey.ac.uk/courses/cs366/MSaint.pdf

Ambler, Scott W. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press, 3rd edition, 2004.

April, J., Better, M., Glover, F., Kelly, J. and Laguna M. Enhancing Business Process Management With Simulation Optimization. In Proceedings of the 38th conference on Winter simulation (Monterey, California, USA, December 3-6, 2006). WSC '06, 642–649.

AtlanMod, AtlantEcore Zoo - An ATL-auto-generated mirror of Atlantic zoo expressed in EMF XMI 2.0, conforming to Ecore, 2008, http://www.emn.fr/x-info/atlanmod/index.php/Ecore

Austin, John L.. How To Do Things With Words. Harvard University Press, 2nd edition, 1975.

Bauer, B. and Odell, J., UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard, in Journal of Engineering Applications of AI, Volume 18, Issue 2, pp 141-157, March, 2005.

Beck, K. 2000. Extreme Programming Explained: Embrace Change. Addison-Wesley Longman, Inc. 190 p.

Bellifemine, F., A. Poggi, and G. Rimassa. Developing Multi-Agent Systems with JADE. In C. Castelfranchi and Y. Lespérance, editors, Intelligent Agents VII. Agent Theories, Architectures, and Languages-7th. International Workshop, ATAL-2000, Boston, MA, USA, July 7–9, 2000, Proceedings, Lecture Notes in Artificial Intelligence. Volume 1986/2001, Springer-Verlag, Berlin, 2001.

Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: Jade Programmer's Guide. JADE 3.6 http://jade.tilab.com/doc/programmersguide.pdf, 18 June 2007

Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., and Munro, M. 2000. Service-based software: the future for flexible software. In Proceedings of the Seventh Asia-Pacific Software Engineering Conference (December 05 - 08, 2000). APSEC. IEEE Computer Society, Washington, DC, 214.

Bergenti, F., Gleizes, M.P. and Zambonelli, F. editors. Methodologies and Software Engineering for Agent Systems. Kluwer, 2004.

Bernon, C., Cossentino, M., Pavon, J.: (2005) Agent Oriented Software Engineering. The Knowledge Engineering Review, Cambridge University Press, 20: pp.99-116.

Beydeda, S., Book, M., Gruhn, V.: Model-Driven Software Development. Springer (2005)

Bloechle, Wendy K. and Schunk, Daniel. MICRO SAINT SHARP SIMULATION SOFTWARE. In Proceedings of the 2003 Winter Simulation Conference, Vol.1, 7-10 Dec. 2003, pp. 182 - 187

Boehm B, "A Spiral Model of Software Development and Enhancement", "Computer", "IEEE", 21(5):61-72, May 1988

Boehm, B. "Get Ready for Agile Methods, with Care," IEEE Computer, vol. 35, no. 1, pp. 64-69, 2002.

Booch, G. 1982. Object-oriented design. Ada Lett. I, 3 (Mar. 1982), 64-76. DOI= http://doi.acm.org/10.1145/989791.989795

Booch G. Object-Oriented Analysis and Design with Applications. Addison Wesley, 1994

Booch, Grady, Rumbaugh, James and Jacobson, Ivar. The Unified Modeling Language Reference Manual, Pearson Higher Education (2nd edition), 2004

Bordini R., Braubach L., Dastani M., El Fallah Seghrouchni A., Gomez-Sanz J.J., Leite J., O'Hare G., Pokahr A., Ricci A., « A Survey of Programming Languages and Platforms for Multi-Agent Systems », Informatica, No. 30, 2006, p. 33-44.

Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W. and Stathis, K.: Crafting the Mind of a PROSOCS Agent. Applied Artificial Intelligence, 20(4-5), April (2006)

Braubach, L., A. Pokahr, and W. Lamersdorf. Extending the Capability Concept for Flexible BDI Agent Modularization. R. Bordini, M. Dastani, J. Dix, and A. El

Fallah Seghrouchni. Proceedings of the Third International Workshop on Programming Multi-Agent Systems (ProMAS'05). 2005. pp.99-114.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A. 2004. TROPOS: An Agent-Oriented Software Development Methodology. J. Auton. Agent. Multi-Ag. 8, 3 (October 2004), 203-236.

Briot J.-P., Meurisse T., Peschanski F., « Une expérience de conception et de composition de comportements d'agents à l'aide de composants », L'Objet, 11(3), 2006.

Budinsky, F., Steinberg, D., Ellersick, R., Merks, E., Brodsky, S.A., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley (2003).

Burrafato, P., Cossentino, M.: Designing a multi-agent solution for a bookstore with the PASSI methodology. Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002). May 2002, Toronto, Ontario, Canada at CAiSE'02

Cervantes, H. and R. S. Hall. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 23-28 May, 2004, pp. 614-623.

Chella, A. Cossentino, M., Sabatucci, L.. Tools and patterns in designing multi-agent systems with PASSI. WSEAS Transactions on Communications, 3(1):352–358, 2004.

Chella, A. Cossentino, M., Sabatucci, L., and Seidita, V., "Agile PASSI: An Agile Process for Designing Agents", International Journal of Computer Science and Eng. Special Issue on "Software Eng. for Multi-Agent Systems", May 2006

Ciancarini P. and Wooldridge M. (eds.) Agent-oriented software engineering. Springer Verlag, Berlin, Germany, 2001.

Cockburn, A. & Highsmith, J. 2001. Agile Software Development: The People Factor. Computer, Vol. 34, No. 11, pp. 131–133

Colombo, R. and Guerra, A. "The Evaluation Method for Software Product", Proc. of the 15th Int. Conf. on Software & Systems Engineering & Applications (ICSSEA 2002), Paris, France, December 3-4, 2002.

Cossentino M. (2005). From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers and Giorgini (2005), chapter IV, pp. 79—106.

Cossentino, M., Gaglio, S., Garro, A. and Seidita, V. 2007. Method fragments for agent design methodologies: from standardisation to research. Int. J. of Agent-Oriented Software Engineering, 1, 1 (April 2007), 91-121.

Dam KH, Winikoff M. Comparing Agent-Oriented Methodologies. In P. Giorgini, B. Henderson-Sellers, and M. Winikoff, editors, Agent-Oriented Information

Systems (AOIS 2003): Revised Selected Papers, pages 78–93. Springer LNAI 3030, 2004, pp. 78-93.

Dastani, M., van Riemsdijk, M.B. and Meyer, J.-J.Ch. Programming multi-agent systems in 3APL. In 2005 Multiagent Systems, Artificial Societies, and Simulated Organizations, 15. Springer-Verlag, 39-67.

David, Alexandre and Deneux, Johann and d'Orso, Julien. A Formal Semantics for UML Statecharts, Uppsala University, 2003, 2003-010, http://www.cs.aau.dk/~adavid/publications/16-TC03-010.pdf

DeLoach, S. A. and Wood, M., "Developing Multiagent Systems with agentTool", Proceedings of the 7th The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL'00), Boston, USA, July, 2000.

Deloach, S.A., Wood, M.F. and Sparkman, C.H. 2001. Multiagent Systems Engineering. Int. J. Softw. Eng. Know. 11, 3 (June 2001), 231-258

Deloach, S.A.: Multiagent systems engineering of organization-based multiagent systems. In: Software Engineering for Multi-Agent Systems IV, LNCS 3914/2006, pages 109-125, Springer Berlin / Heidelberg, 2006.

Depke R., Heckel R. and Kuster J.M. 2002. Formal agent-oriented modeling with UML and graph transformation. J. Sci. Comput. Program. 44, 2, 229-252.

Dunn-Davies, H.R., Cunningham, R.J. and Paurobally, S. Propositional Statecharts for Agent Interaction Protocols. In 2005 Electronic Notes in Theoretical Computer Science, 134. Elsevier, 55-75.

Eclipse. An open source integrated development environment (IDE), http://www.eclipse.org/

Efftinge, Sven and Völter, Markus. oAW xText: A framework for textual DSLs. In Eclipse Summit 2006 Workshop: Modeling Symposium, 2006. http://www.eclipsecon.org/summiteurope2006/presentations/ESE2006-EclipseModelingSymposium12_xTextFramework.pdf

FIPA TC Agent Management, FIPA Agent Management Specification, Document number SC00023K, Foundation for Intelligent Physical Agents, March 2004, http://www.fipa.org/specs/fipa00023/

FIPA TC Communication, FIPA Request Interaction Protocol Specification, Document number SC00026H, Foundation for Intelligent Physical Agents, December 2002a. URL: http://www.fipa.org/specs/ fipa00026/

FIPA TC Communication, FIPA Communicative Act Library Specification, Document number SC00037J, Foundation for Intelligent Physical Agents, December 2002b. URL: http://www.fipa.org/specs/fipa00037/

FIPA TC Communication, FIPA ACL Message Structure Specification, Document number SC00061G, Foundation for Intelligent Physical Agents, December 2002c. URL: http://www.fipa.org/specs/fipa00061/

Fischer, M. J. and Ladner, R. E. 1977. Propositional modal logic of programs. In Proceedings of the Ninth Annual ACM Symposium on theory of Computing (Boulder, Colorado, United States, May 04 - 04, 1977). STOC '77. ACM, New York, NY, 286-294. DOI= http://doi.acm.org/10.1145/800105.803418

Fornara, N. and Colombetti, M. 2003. Defining interaction protocols using a commitment-based agent communication language. In *Proceedings of the Second international Joint Conference on Autonomous Agents and Multiagent Systems* (Melbourne, Australia, July 14 - 18, 2003). AAMAS '03. ACM, New York, NY, 520-527. DOI= http://doi.acm.org/10.1145/860575.860659

Fowler, M. and J. Highsmith, "The Agile Manifesto," in Software Development, August 2001, pp. 28-32.

Garcia, J., Laguna, M., Carvajal, Y. and Baixauli B.: Requirements variability support through MDD and graph transformation. In International Workshop on Graph and Model Transformation, pp. 161--173. Tallinn, Estonia (2006)

García-Magariño, Iván, Gómez-Sanz, Jorge J. and Fuentes-Fernández, Rubén. Model Transformations for Improving Multi-agent Systems Development in INGENIAS. The 10th International Workshop on Agent-Oriented Software Engineering AOSE'09 May 11, 2009, Budapest Hungary

Garro, A., Turci, P., Huget, M.P.: Meta-Model Sources: Gaia. FIPA Methodology Technical Committee, Foundation for Intelligent Physical Agents, http://www.fipa.org/ , 2004

Gerber, A., Raymond, K.: Mof to emf: there and back again. In: eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, New York, NY, USA, ACM Press (2003) 60-64

Ghezzi, C., Jazayeri, M., Mandrioli, D.: Fundamentals of Software Engineering. Prentice Hall, ISBN: 0133056996, 2002

Giorgini Paolo, Kolp Manuel, Mylopoulos John, Castro Jaelson. Tropos: A Requirements-Driven Methodology for Agent-Oriented Software. In Henderson-Sellers B. and Giorgini P. (eds) Agent-Oriented Methodologies. Idea Group Publishing. 2005, pp. 20-45

Hahn, C., Madrigal-Mora, C., Fischer, K. A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems, Volume 18, Issue 2, April 2009, Pages 239-266

Harel, D. and Naamad, A. 1996. The STATEMATE Semantics of Statecharts. ACM T. Softw. Eng. Meth. 5, 4 (October 1996), 293-333.

Harel D. and Kugler, H. “The RHAPSODY Semantics of Statecharts (Or on the Executable Core of the UML)”, Integration of Software Specification Techniques for Application in Engineering, LNCS 3147, Springer-Verlag Berlin Heidelberg, 2004, pp. 325–354

Henderson-Sellers B. and Giorgini P. 2005 Agent-Oriented Methodologies. Idea Group Publishing.

Hirsch, M.: Making RUP agile. In: Conference on Object Oriented Programming Systems Languages and Applications. ACM Press New York, NY, USA, 2002.

IEEE Standard Glossary of Software Engineering Terminology. IEEE std 610.12-1990, 1990

Iglesias, C. A., Garijo, M., González, J. C., & Velasco, J. R. (1998). Analysis and design of multiagent systems using MAS-CommonKADS. In N. Callaos & M. Torres (Eds.), Intelligent agents IV: Agent theories, architectures and languages, LNAI 1365 (pp. 313-326). Berlin: Springer-Verlag.

Jacyntho, M. D., Schwabe, D., and Rossi, G. (2002). A software architecture for structuring complex web applications. Journal of Web Engineering, 2(1-2), pp. 37-60.

Jayatilleke, G.B., Padgham, L. and Winikoff, M. 2005. A model driven component-based development framework for agents. Int. J. Comput. Syst. Sci. Eng. 20, 4 (July), 273-282.

Jennings, N.R.: Agent-Oriented Software Engineering. In: Proceedings of the 9th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW’99). LNCS, 1647, Springer-Verlag (1999)

Jouault F., Bézivin J., KM3: A DSL for Metamodel Specification. In Formal Methods for Open Object-Based Distributed Systems, Proceedings of the 8th IFIP WG 6.1 International Conference, (FMOODS 2006), Bologna, Italy, June 14-16, 2006. pp. 171-185.

Jouault, F. and Kurtev, I. On the Architectural Alignment of ATL and QVT. In Proceedings of the 2006 ACM Symposium on Applied Computing (Dijon, France, April 23-27, 2006). SAC 06. ACM Press, 1188-1195 (2006a)

Jouault, F. and Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference. Volume 3844 of Lecture Notes in Computer Science., Springer-Verlag 128–138 (2006b)

Kakas A, Moraitis P, Argumentation based decision making for autonomous agents. In Proc of the 2nd Int Conf on Auton Agents and Multi-Agent Syst, Melbourne, Australia, July 14-18, 2003

Karacapilidis, N., Moraitis P.,: Intelligent Agents for an Artificial Market System. Proc. fifth International Conference on Autonomous Agents (AGENTS'01), pp. 592-599, Montreal, Canada (2001)

Klatt, B., Xpand : A Closer Look at the Model2Text Transformation Language, http://bar54.de/benjamin.klatt-Xpand.pdf , 06 July 2007.

Kleppe, A., Warmer, S. and Bast, W. 2003 MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley.

Klucsh M., Sycara K.: Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In Omicini et al. (editor), Coordination of Internet Agents, Springer, 2001

Knublauch, H. ''Extreme programming of multi-agent systems,'' in {tiProceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems}, ACM Press: Bologna (I), 2002, pp. 704-711.

Knuth, Donald E.: The art of computer programming, volume 1 (3rd ed.): fundamental algorithms, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1997

Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The Epsilon Object Language (EOL). In: Rensink, A.,Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 128–142. Springer, Heidelberg (2006)

König R.. State-based modeling method for multiagent conversation protocols and decision activities. In R. Kowalczyk, J. P. M¨uller, H. Tianfield, and R. Unland, editors, Agent Technologies, Infrastructures, Tools, and Applications for E-Services, volume 2592 of Lecture Notes in Computer Science, pages 151–166. Springer, 2003. http://www.springerlink.com/content/8yqqm6b2w6eprnqx/

Kruchten, P. The Rational Unified Process, An Introduction. Addison-Wesley, Second Edition, 2000

Labrou, Y.; Finin, T.; Yun Peng, "Agent communication languages: the current landscape," Intelligent Systems and their Applications, IEEE , vol.14, no.2, pp.45-52, Mar/Apr 1999, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=757631&isnumber=16422

Lamsweerde, A. V. Goal-Oriented Requirements Engineering: A Guided Tour, Proceedings RE'01 - 5th IEEE International Symposium on Requirements Engineering, Toronto, August, 2001, pp. 249-263. Available at: http://www.isys.ucl.ac.be/staff/stephane/EGMI2110Slide/Avl01.pdf

Langlois, B., Jitia, C.E., Jouenne, E.. DSL Classification. In 7th OOPSLA Workshop on Domain-Specific Modeling, 2007. http://www.dsmforum.org/events/DSM07/papers/langlois_jitia.pdf

Larman, C.. Agile and Iterative Development: A Manager's Guide. Addison-Wesley Pub Co; 1st edition, 2003

Lonchamp Jacques: A Structured Conceptual and Terminological Framework for Software Process Engineering. Proceedings of the Second International

Conference on the Software Process: Continuous Software Process Improvement, Berlin, Germany, February 25-26, 1993. IEEE Computer Society, 1993, ISBN 0-8186-3600-9, pp. 41-53

Matsatsinis, N., Moraitis, P., Psomatakis, V. and Spanoudakis, N. “An Agent-Based System for Products Penetration Strategy Selection”, Applied Artificial Intelligence Journal, Taylor & Francis, Vol. 17, No. 10, 2003, pp. 901-925.

Mazouzi, H., Seghrouchni, A. E., and Haddad, S. 2002. Open protocol design for complex interactions in multi-agent systems. In Proceedings of the First international Joint Conference on Autonomous Agents and Multiagent Systems: Part 2 (Bologna, Italy, July 15 - 19, 2002). AAMAS '02. ACM, New York, NY, 517-526. DOI= http://doi.acm.org/10.1145/544862.544866

Meyer, B.. Object-Oriented Software Construction, Prentice Hall PTR, ISBN 0136291554, 1997

Mikk, E., Lakhnech, Y., Petersohn, C. and Siegel, M.. On formal semantics of Statecharts as supported by STATEMATE. In Second BCS-FACS Northern Formal Methods Workshop. Springer-Verlag, 1997, http://reference.kfupm.edu.sa/content/o/n/on_formal_semantics_of_statecharts_as_su_123245.pdf

Moore, Scott A. and Kimbrough, Steven O.. Message management systems at work: Prototypes for business communication. Journal of Organizational Computing, 5(2):83-100, 1995.

Moore, Scott A.. A foundation for flexible automated electronic commerce. Working paper at the University of Michigan Business School no. 99-011, May 1999. http://deepblue.lib.umich.edu/bitstream/2027.42/35926/2/b2014129.0001.001.pdf

Moore, S.A. “On conversation policies and the need for exceptions”, Issues in Agent Communication, Lecture Notes in Artificial Intelligence 1916, Springer, 2000, pp. 144–159.

Moraitis P., "Multi-Agent Systems Paradigm and Distributed Decision Making", PhD Thesis, Paris Dauphine University, December 1994

Moraitis, P.: Decision Theoretic and Logic Based Agents for Multi-Agent Systems. Habilitation for Research Supervising, University Paris-Dauphine, France (2002)

Moraitis, P., Petraki, E., Spanoudakis, N.: Engineering JADE Agents with the Gaia Methodology. R. Kowalszyk, J. Muller, H. Tianfield, R. Unland (eds), Lecture Notes in Computer Science (LNCS), vol. 2592: “Agent Technologies, Infrastructures, Tools, and Applications for e-Services”, Springer-Verlag, 2003a, pp 77-91

Moraitis, P., Petraki, E. and Spanoudakis, N.. Providing Advanced, Personalised Infomobility Services Using Agent Technology. In: Proc. of the 23rd SGAI

International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI2003), Peterhouse College, Cambridge, UK, December 15-17, 2003b

Moraitis, P., Petraki, E. and Spanoudakis, N.. An Agent-based System for Infomobility Services. In: Proceedings of the third European Workshop on Multi-Agent Systems (EUMAS2005), Brussels, Belgium, December 7 - 8, 2005

Moraitis, P. and Spanoudakis N. 2006. The Gaia2JADE Process for Multi-Agent Systems Development. J. Appl. Artif. Intell. 20, 2-4 (February-April 2006), 251-273.

Moraitis, P., Spanoudakis N.: Argumentation-based Agent Interaction in an Ambient Intelligence Context. IEEE Intelligent Systems, 22(6), pp. 84-93 (2007)

O'Hare, G. M. P., & Jennings, N. R. (Eds.). (1996) . Foundations of distributed artificial intelligence. New York: John Wiley & Sons, Inc.

Object and Reference Model Subcommittee (ORMSC) of the OMG Architecture Board: A Proposal for an MDA Foundation Model, white paper OMG-ORMSC/05-08-01, http://www.omg.org/cgi-bin/doc?ormsc/05-08-01 (2005)

Object Management Group. Meta Object Facility (MOF) Core Specification (06-01-01), January 2001. http://www.omg.org/spec/MOF/2.0/PDF/

Object Management Group: Software Process Engineering Metamodel Specification. OMG, 2002, http://www.omg.org

Object Management Group: Object Constraint Language (OCL), OMG Document ptc/03-10-14 (2003)

Object Management Group. Human-Usable Textual Notation V1.0 (04-08-01), August 2004. http://www.omg.org/docs/formal/04-08-01.pdf

Object Management Group: Revised Submission for MOF 2.0 Query/View/Transformations RFP, OMG Document ad/2005-07-01 (2005)

Odell, J. Objects and agents compared. Journal of Object Computing, 1(1), May 2002.

Odell J., Parunak H. V. D. and Bauer B. Extending UML for Agents. In Proc. of the Agent-Oriented Information Systems (AOIS) Workshop at the 17th National conference on Artificial Intelligence (AAAI), 2000.

Odell, J., Parunak, H. V. D. and Bauer, B. Representing Agent Interaction Protocols in UML. In Agent-Oriented Software Engineering, LNCS 1957/2001. Springer, 201-218.

OSGi alliance. A worldwide consortium of technology innovators defining a component integration platform, http://www.osgi.org

Padgham, L. and Winikoff, M. Prometheus: A Methodology for Developing Intelligent Agents. In G. Goos, J. Hartmanis, and J. van Leeuwen (eds) Agent-Oriented

Software Engineering III, Lecture Notes in Computer Science, Volume 2585/2003, Springer Berlin / Heidelberg, ISBN 978-3-540-00713-5, DOI 10.1007/3-540-36540-0, pp. 174-185

Padgham, L. and Winikoff, M. 2004 Developing Intelligent Agent Systems: A Practical Guide. Wiley.

Padgham, L. and Winikoff, M. Prometheus: A Practical Agent-Oriented Methodology. In Henderson-Sellers B. and Giorgini P. (eds) Agent-Oriented Methodologies. Idea Group Publishing. 2005

Paurobally, S., Cunningham, R. and Jennings, N.R. Developing agent interaction protocols using graphical and logical methodologies. In 2004 Programming Multi-Agent Systems, LNCS 3067/2004. Springer, 149-168.

Pavón J., and Gómez-Sanz J. (2003). Agent-Oriented Software Engineering with INGENIAS. In: Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03), Lecture Notes in Computer Science 2691, Springer Verlag, pp. 394-403.

Pavón, J., Gómez-Sanz, J.J. & Fuentes, R.: The INGENIAS Methodology and Tools. In: Henderson-Sellers, B. and Giorgini, P., editors: Agent-Oriented Methodologies. Idea Group Publishing, pp. 236-276 (2005)

Perini, A. and Susi, A. Automating Model Transformations in Agent-Oriented Modeling. In 2006 Agent-Oriented Software Engineering VI, LNCS 3950. Springer, 167-178.

Petri, Carl Adam; Reisig, Wolfgang. "Petri net". Scholarpedia 3(4):6477. http://www.scholarpedia.org/article/Petri_net

Pikkarainen, Minna (2008) Towards a Framework for Improving Software Development Process Mediated with CMMI Goals and Agile Practices. VTT Publications 695. Espoo: VTT Technical Research Centre of Finland. ISBN 978-951-38-7122-2, URI: http://www.vtt.fi/inf/pdf/publications/2008/P695.pdf

Protégé. A free, open source ontology editor and knowledge-base framework, http://protege.stanford.edu/

Rana, O. F. and Stout, K. 2000. What is scalability in multi-agent systems?. In Proceedings of the Fourth international Conference on Autonomous Agents (Barcelona, Spain, June 03 - 07, 2000). AGENTS '00. ACM, New York, NY, 56-63. DOI= http://doi.acm.org/10.1145/336595.337033

Ricordel, P. and Demazeau, Y. (2002). Volcano, a vowels-oriented multi-agent platform. In From Theory to Practice in Multi-Agent Systems, Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001), volume 2296 of Lecture Notes in Computer Science, pages 253-262. Springer.

Rose, L.M., Paige, R.F., Kolovos, D.S. and Polack, F.. Constructing models with the Human-Usable Textual Notation. In 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS), volume 5301 of LNCS, pages 249-263. Springer, 2008.

Rosen H. Kenneth. Discreet Mathematics and its Applications. Forth edition, McGraw Hill, 1999

Royce, Winston W. Managing the Development of Large Software Systems: Concepts and Techniques. In: Technical Papers of Western Electronic Show and Convention (WesCon) August 25-28, 1970, Los Angeles, USA.

Russel, Stuart and Norvig, Peter. Artificial Intelligence a Modern Approach. Second Edition, Prentice Hall, 2003

Schwaber, K. & Beedle, M. 2002. Agile Software Development with Scrum. Prentice-Hall. Upper Saddle River, NJ. 0-13-067634-9 158.

Sendall, S. and Kozaczynski, W. 2003. Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Softw. 20, 5 (Sep. 2003), 42-45. DOI= http://dx.doi.org/10.1109/MS.2003.1231150

Smith, Reid G. and Davis, Randall. Frameworks for Cooperation in Distributed Problem Solving. In: IEEE Transactions on Systems, Man and Cybernetics, Volume: 11, Issue 1, Jan. 1981, pp. 61-70, ISSN: 0018-9472 http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&arnumber=4308579&isnumber=4308570

Sowmya, S. R. and Ramesh, S. - Extending Statecharts with Temporal Logic, IEEE Transactions on Software Engineering, Vol. 24, No. 3, March 1998.

Spanoudakis N., Zangherati S. and Lazaro Ramos J.P.: Agent Platform Architecture. ASK-IT project internal deliverable ID3.1.1, 2005, http://www.ask-it.org

Spanoudakis N., Moraitis, P.: Engineering a Brokering Framework for Providing Semantic Services to Agents on Lightweight Devices. In: Proc. ECAI'06 Workshop on Context and Ontologies: Theory, Practice and Applications (C&O'06), Riva del Garda, Italy, 2006a

Spanoudakis N., Moraitis, P.: Agent Communication Protocol. ASK-IT project internal deliverable ID3.1.2, 2006b, http://www.ask-it.org

Spanoudakis, N. and Moraitis, P.. An Ambient Intelligence Application Integrating Agent and Service-Oriented Technologies. In: Proceedings of the 27th SGAI International Conference on Artificial Intelligence (AI2007), Peterhouse College, Cambridge, UK, December 10-12, 2007a

Spanoudakis N., Moraitis, P.: The Agent Systems Methodology (ASEME): A Preliminary Report. In: Proceedings of the 5th European Workshop on Multi-Agent Systems (EUMAS'07), Hammamet, Tunisia, December 13 - 14, 2007b

Spanoudakis N. and Pendaraki K.. A Tool for Portfolio Generation Using an Argumentation Based Decision Making Framework. In Proc of the Annu IEEE Int Conf on Tools with Artif Intell, Patras, Greece, October 29-31, 2007c

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), Springer, Lecture notes in Computer Science (LNCS), Volume 5253/2008, Varna, Bulgaria, September 4-6, 2008a

Spanoudakis, N. and Moraitis, P.. An Agent Modeling Language Implementing Protocols through Capabilities. In: Proceedings of The 2008 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-08) , Sydney, Australia, December 9-12, 2008b

Spanoudakis N., Moraitis, P.: An Autonomous Agent Application for Product Pricing. In: Proceedings of the 6th European Workshop on Multi-Agent Systems (EUMAS'08), Bath, UK, December 18-19, 2008c

Spanoudakis N., Moraitis, P.: Automated Product Pricing Using Argumentation. In: Proceedings of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI 2009) Thessaloniki, Greece, April 23-25, 2009

Spivey, J. Michael. The Z Notation: a Reference Manual. Prentice Hall, 1989.

Stevens W., G. Myers, L. Constantine, "Structured Design", IBM Systems Journal, 13 (2), 115-139, 1974.

Sturm, Arnon and Shehory, Onn. A comparative evaluation of agent oriented methodologies. In Bergenti et al. (2004), chapter 7, pp. 127-149

Susi, A., Perini, A., Giorgini, P. and Mylopoulos, J.. The Tropos Metamodel and its Use. Informatica, 29(4):401–408, 2005.

SWI-Prolog. A Free Software Prolog environment, http://www.swi-prolog.org/

Szyperski, C., “Component Software: beyond object-oriented programming”, ACM Press/Addison-Wesley Publishing Co., 1997.

Tolvanen, J. P. 1998. Incremental Method Engineering with Modeling tools. Doctoral Thesis. University of Jyväskylä. 301 p.

Toulis, P., Tzovaras, D., Spanoudakis, N. “MARKET-MINER Project Exploitation Plan”, MARKET-MINER Project, Deliverable Π6.1 (in Greek language), Singular Logic S.A., Greece, February 2007a.

Toulis, P., Tzovaras, D., Pantelopoulos, S. “MARKET-MINER System Evaluation Report”, MARKET-MINER Project, Deliverable Π5.1 (in Greek language), Singular Logic S.A., Greece, February 2007b.

Trencansky, I. and Cervenka, R.: Agent Modeling Language (AML): A comprehensive approach to modeling MAS. Informatica 29(4), pp. 391–400 (2005)

UML. Unified Modeling Language Specification. ISO 19501:2005

Wąsowski, A., and Sestoft, P. On the formal semantics of visualSTATE statecharts. Tech. Rep. TR-2002-19, IT University of Copenhagen, Sept. 2002. http://www.it-c.dk/~wasowski/papers/vsfsem.pdf

Wąsowski A.: Code Generation and Model Driven Development for Constrained Embedded Software. PhD dissertation. IT University of Copenhagen (2005) https://www.itu.dk/people/wasowski/papers/wasowski-dissertation-20050909.pdf

Weiss, Michael. Patterns for Motivating an Agent-Based Approach. In G. Goos, J. Hartmanis, and J. van Leeuwen (eds) Conceptual Modeling for Novel Application Domains, Lecture Notes in Computer Science, Volume 2814/2003, Springer Berlin / Heidelberg, 2003

Winikoff, M. Jack Intelligent Agents: An Industrial Strength Platform. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix and Amal El Fallah Seghrouchni (eds), Multi-Agent Programming Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, Volume 15, ISBN 978-0-387-24568-3 (Print) 978-0-387-26350-2 (Online), DOI 10.1007/b137449, Springer US, 2005, pp. 175-193

Winikoff, M.. JACK^{TM} intelligent agents: An industrial strength platform. Chapter 7 in R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. Multi-Agent Programming: Languages, Platforms and Applications. Number 15 in Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005, pages 175–193.

Wirth, Niklaus. Extended Backus-Naur Form (EBNF), 1996. ISO/IEC 14977:1996(E).

Wooldridge, M. & Jennings, N. (1995), “Intelligent Agents: Theory and Practice”, The Knowledge Engineering Review 10 (2), 115-152.

Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems Vol. 3. No. 3 (2000) 285-312

Yamamoto, G. and Nakamura, Y. 1999. Architecture and performance evaluation of a massive multi-agent system. In Proceedings of the Third Annual Conference on Autonomous Agents (Seattle, Washington, United States). O. Etzioni, J. P. Müller, and J. M. Bradshaw, Eds. AGENTS '99. ACM, New York, NY, 319-325. DOI= http://doi.acm.org/10.1145/301136.301219

Young DA - 1992 - Object-oriented programming with C++ and OSF/Motif, Prentice-Hall, Inc. Upper Saddle River, NJ, USA

Zambonelli, F., Jennings, N.R. and Wooldridge, M. 2003. Developing multiagent systems: the Gaia Methodology. ACM T. Softw. Eng. Meth. 12, 3 (July 2003), 317-370.

Zhu, H. and Shan, L. Caste-Centric Modeling of Multi-Agent Systems: The CAMLE Modeling Language and Automated Tools. In Beydeda, S. and Gruhn, V. (eds.) 2005 Model-driven Software Development. Springer, 57-89.

Annex 2.

Abbreviations

A

AC	Active Configuration (§2.1.3)
ACL	Agent Communication Language (§2.2.9)
AID	Agent Identifier (§4.5)
AMOLA	Agent Modeling Language (§Chapter 3)
ANML	Agent Negotiation Meta-Language (§2.2.9.3)
AOSE	Agent-Oriented Software Engineering (§2.2)
API	Application Programming Interface
ASEME	Agent Systems Engineering Methodology (§Chapter 4)
ATL	ATLAS Transformation Language (§2.1.4.4)
AUML	Agent UML (§2.2.3)
AVSP	Added-Value Service Provider (§3.2.1)

B

BR	Broker Agent (§3.2.1)

C

CASE	Computer-Aided Software Engineering, http://en.wikipedia.org/wiki/Computer-aided_software_engineering
CIM	Computation Independent Model (§2.1.4.4)
CIS	Conversation Identification String (§4.8)
CP	Conversation Policy (§2.2.9.2)
CPN	Colored Petri Nets (§2.2.9.3)
CT	Compound Transition (§2.1.3)

E

EBNF	Extended Backus–Naur Form (§3.3.3)
EMF	Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/

F	
FIPA	Foundation for Intelligent Physical Agents, http://fipa.org
FLBC	Formal Language for Business Communication (§2.2.9.2), http://www-personal.umich.edu/~samoore/research/flbc/
FSM	Finite State Machine (§2.2.9.3)
H	
HUTN	Human-Usable Textual Notation (§5.2.2), http://www.omg.org/technology/documents/formal/hutn.htm
I	
IAC	Intra-Agent Control (§3.4)
IDE	Integrated Development Environment
ISV	Independent Software Vendor
J	
JADE	Java Agent Development Environment, http://jade.tilab.com/
JDBC	Java DataBase Connectivity, http://java.sun.com/javase/technologies/database/
JET	Java Emitter Templates (§5.2.3)
JSP	JavaServer Pages, http://java.sun.com/products/jsp/
K	
KM3	Kernel MetaMetaModel (§2.1.4.4)
M	
M2M	Model to Model transformation (Chapter 5)
M2T	Model to Text transformation (Chapter 5)
MAS	Multi-Agent System
MDA	Model-Driven Architecture (§2.1.4.4)
MDE	Model-Driven Engineering (§2.1.4.4)
MIPA	Market-Miner Project Product Pricing Agent (§4.9.4)
MOF	Meta-Object Facility, http://www.omg.org/technology/documents/formal/mof.htm
O	
oAW	openArchitectureWare, a platform for model-driven software development (§5.2.3), http://www.openarchitectureware.org/
OMG	Object Management Group, http://www.omg.org
OOD	Object Oriented Development (or Design, §2.1.2)
OOP	Object Oriented Programming (§2.1.2)
OSGi	Open Services Gateway initiative, http://www.osgi.org
P	
PA	Personal Assistant (§3.2.1)
PDL	Propositional Dynamic Logic (§2.2.9.3)
PIM	Platform Independent Model (§2.1.4.4)
PSM	Platform Specific Model (§2.1.4.4)
Q	
QVT	Query/Views Transformations (§2.1.4.4)
R	
RPG	Requirements per Goal document (§3.2.1)
RUP	Rational Unified Process (§2.1.2.2)

S

SAG	System Actors and Goals (§3.2)
SP	Service Provider (§3.3.2)
SR	Service Requester (§3.3.2)
SRM	System Roles Model (§3.3)
STD	State Transition Diagram (§2.2.9.3)
SUC	System Use Cases (§3.3)

T

T2M	Text to Model transformation (§5.2.2)
T2T	Text to Text transformation (Chapter 5)

U

UML	Unified Modeling Language (§2.1.2.1)

Annex 3.

The AMOLA Metamodels

Listing 20. The SAG metamodel definition in XML format (SAG.ecore file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="SAG"
nsURI="http://mi.parisdescartes.fr/ASEME/metamodels/SAG" nsPrefix="SAG">
  <eClassifiers xsi:type="ecore:EClass" name="Actor">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="my_goal" upperBound="-1"
eType="#//Goal" eOpposite="#//Goal/depender" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Goal">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="depender" lowerBound="1"
eType="#//Actor" eOpposite="#//Actor/my_goal" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="dependee" upperBound="-1"
eType="#//Actor" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="requirements"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  </eClassifiers>
</ecore:EPackage>
```

Listing 21. The SUC metamodel definition in XML format (SUC.ecore file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="SUC"
nsURI="http://mi.parisdescartes.fr/ASEME/metamodels/SUC" nsPrefix="SUC">
  <eClassifiers xsi:type="ecore:EClass" name="UseCase">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="interacter" ordered="false"
upperBound="-1" eType="#//Role" eOpposite="#//Role/interacts_with" />
```

```
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="specified_by"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="include" ordered="false"
upperBound="-1" eType="#//UseCase" eOpposite="#//UseCase/included_by" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="included_by" ordered="false"
upperBound="-1" eType="#//UseCase" eOpposite="#//UseCase/include" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Role">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="interacts_with"
ordered="false" upperBound="-1" eType="#//UseCase" eOpposite="#//UseCase/interacter"
/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="HumanRole" eSuperTypes="#//Role" />
  <eClassifiers xsi:type="ecore:EClass" name="SystemRole" eSuperTypes="#//Role" />
</ecore:EPackage>
```

Listing 22. The SRM metamodel definition in XML format (SRM.ecore file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="SRM"
nsURI="http://mi.parisdescartes.fr/ASEME/metamodels/SRM" nsPrefix="SRM">
  <eClassifiers xsi:type="ecore:EClass" name="Role">
  <eStructuralFeatures xsi:type="ecore:EReference" name="activities" ordered="false"
lowerBound="1" upperBound="-1" eType="#//Activity" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="protocols" ordered="false"
upperBound="-1" eType="#//Protocol" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="liveness" ordered="false"
lowerBound="1" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="capabilities" ordered="false"
upperBound="-1" eType="#//Capability" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Activity">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" ordered="false"
unique="false" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="functionality"
ordered="false" unique="false" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Protocol">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" ordered="false"
unique="false" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="participant" ordered="false"
unique="false" lowerBound="1" upperBound="-1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Capability">
  <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" ordered="false"
unique="false" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="activities" upperBound="-1"
eType="#//Activity" />
  </eClassifiers>
</ecore:EPackage>
```

Listing 23. The IAC metamodel definition in XML format (IAC.ecore file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" xmi:id="_3GNeoVAnEd64JYB58_U7Wg"
name="IAC" nsURI="http://mi.parisdescartes.fr/ASEME/metamodels/IAC" nsPrefix="IAC">
  <eClassifiers xsi:type="ecore:EClass" xmi:id="_3GNeolAnEd64JYB58_U7Wg" name="Node">
```

```
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFsFAnEd64JYB58_U7Wg"
name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFslAnEd64JYB58_U7Wg"
name="type" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFtFAnEd64JYB58_U7Wg"
name="label" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"
/>
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFtlAnEd64JYB58_U7Wg"
name="activity" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" xmi:id="_3GOFuFAnEd64JYB58_U7Wg"
name="variables" upperBound="-1" eType="#_3GOFu1AnEd64JYB58_U7Wg" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" xmi:id="_3GOFuVAnEd64JYB58_U7Wg" name="root"
eSuperTypes="#_3GNeolAnEd64JYB58_U7Wg" />
  <eClassifiers xsi:type="ecore:EClass" xmi:id="_3GOFu1AnEd64JYB58_U7Wg"
name="Variable">
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFvFAnEd64JYB58_U7Wg"
name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFvlAnEd64JYB58_U7Wg"
name="type" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" xmi:id="_3GOFwFAnEd64JYB58_U7Wg"
name="Transition">
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFwVAnEd64JYB58_U7Wg"
name="TE" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" xmi:id="_3GOFw1AnEd64JYB58_U7Wg"
name="source" lowerBound="1" eType="#_3GNeolAnEd64JYB58_U7Wg" />
  <eStructuralFeatures xsi:type="ecore:EReference" xmi:id="_3GOFxVAnEd64JYB58_U7Wg"
name="target" lowerBound="1" eType="#_3GNeolAnEd64JYB58_U7Wg" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFx1AnEd64JYB58_U7Wg"
name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" xmi:id="_3GOFyVAnEd64JYB58_U7Wg" name="Model">
  <eStructuralFeatures xsi:type="ecore:EReference" xmi:id="_3GOFylAnEd64JYB58_U7Wg"
name="nodes" upperBound="-1" eType="#_3GNeolAnEd64JYB58_U7Wg" />
  <eStructuralFeatures xsi:type="ecore:EReference" xmi:id="_3GOFzFAnEd64JYB58_U7Wg"
name="transitions" upperBound="-1" eType="#_3GOFwFAnEd64JYB58_U7Wg" />
  <eStructuralFeatures xsi:type="ecore:EAttribute" xmi:id="_3GOFzlAnEd64JYB58_U7Wg"
name="name" eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString" />
  <eStructuralFeatures xsi:type="ecore:EReference" name="variables" upperBound="-1"
eType="#_3GOFu1AnEd64JYB58_U7Wg" />
  </eClassifiers>
</ecore:EPackage>
```

Annex 4.

The SRM2IAC transformation project files

Listing 24. The main java file implementing the Liveness2HUTN transformation (Liveness2IAC_HUTN.java)

```
package fr.parisdescartes.mi.aseme.t2m.srm2iac;

import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.StringTokenizer;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Liveness2IAC_HUTN {

  String liveness = null;
  Hashtable<String, Node> nodes = new Hashtable<String, Node>();
  Hashtable<String, Variable> variables = new Hashtable<String, Variable>();
  Hashtable<String, Transition> transitions = new Hashtable<String, Transition>();
  Hashtable<String, String> formulas = new Hashtable<String, String>();

  /**
   * @param args
   */
  public static void main(String[] args) {
    // TODO Auto-generated method stub
    Liveness2IAC_HUTN liveness2IAC = new Liveness2IAC_HUTN();
    liveness2IAC.liveness = args[0];
    liveness2IAC.liveness=liveness2IAC.liveness.replaceAll(" ", "");
    liveness2IAC.transform();
  }
```

```
  public void transform() {
    StringTokenizer line = new StringTokenizer(this.liveness, "\n");
    while (line.hasMoreTokens()) {
      String tmp = line.nextToken();
      StringTokenizer formulaElements = new StringTokenizer(tmp, "=");
      String leftHandSide = formulaElements.nextToken();
      String formula = formulaElements.nextToken();
      formulas.put(leftHandSide, formula);
    }
    line = new StringTokenizer(liveness, "\n");
    StringTokenizer formulaElements = new StringTokenizer(line.nextToken(),
        "=");
    String leftHandSide = formulaElements.nextToken();
    line = null;
    formulaElements = null;
    Node root = new Node();
    root.setLabel("0");
    root.setName(leftHandSide);
    this.nodes.put(root.getLabel(), root);
    // call the createStatechart recursive process
    this.createStatechart(formulas.get(leftHandSide),
        root.getLabel());
    //create model
    Model m = new Model();
    for (Enumeration<String> iterator = nodes.keys(); iterator.hasMoreElements();) {
      m.addNode(iterator.nextElement());
    }
    for (Enumeration<String> iterator = transitions.keys();
iterator.hasMoreElements();) {
      m.addTransition(iterator.nextElement());
    }
    // create output
    try {
      PrintWriter out = new PrintWriter(new FileOutputStream(new File(
          "IAC.hutn")));
      out
          .write(new String(
              "@Spec{\n"
                  + " Metamodel \"IAC\"{\n"
                  + "  nsUri: \"http://mi.parisdescartes.fr/ASEME/metamodels/IAC\"\n"
                  + " }\n" + "}\n" + "IAC{\n"));
      for (Enumeration<Node> enumerator = nodes.elements(); enumerator
          .hasMoreElements();) {
        out.write(enumerator.nextElement().toHutnString());
      }
      for (Enumeration<Variable> enumerator = variables.elements(); enumerator
          .hasMoreElements();) {
        out.write(enumerator.nextElement().toHutnString());
      }
      for (Enumeration<Transition> enumerator = transitions.elements(); enumerator
          .hasMoreElements();) {
        out.write(enumerator.nextElement().toHutnString());
      }
      out.write(m.toHutnString());
      out.write(new String("}\n"));
      out.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void createStatechart(String expression, String father) {
    // this integer will be used for selecting the connector for getting the
    // terms after the if section
    int expressionType = 0;
    // pattern for parallelExpression : expressionType=1
    Pattern patternParallelExpression = Pattern

  .compile("(((\\(.+\\))|([\\w&&[^()]])+)(ω?)\\|\\|)+((\\(.+\\))|([\\w&&[^()]])+)(ω?)"
);
    Matcher parallelMatcher = patternParallelExpression.matcher(expression);
    // pattern for orExpression : expressionType=2
    Pattern patternOrExpression = Pattern

  .compile("(((\\(.+\\))|([\\w&&[^()]])+)(ω?)\\|)+((\\(.+\\))|([\\w&&[^()]])+)(ω?)");
```

```
    Matcher orMatcher = patternOrExpression.matcher(expression);
    // pattern for sequentialExpression : expressionType=3
    Pattern patternSequentialExpression = Pattern

  .compile("(((\\(.+\\))|([\\w&&[^()]])+)(ω?)\\.)+((\\(.+\\))|([\\w&&[^()]])+)(ω?)");
    Matcher sequentialMatcher = patternSequentialExpression
        .matcher(expression);
    Node tmpNode;
    Transition tmpTransition;
    if (sequentialMatcher.find()
        && (sequentialMatcher.group().length() == expression.length())) {
      expressionType = 3;
      System.out.print("a sequential expression processed: " + expression
          + "\n");
      nodes.get(father).setType(Node.TYPE_OR);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".1");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_START);
      nodes.put(tmpNode.getLabel(), tmpNode);
      int k = 2;
      for (Iterator<String> iterator = this.findTermsInExpression(
          expression, ".").iterator(); iterator.hasNext();) {
        String term = iterator.next();
        tmpNode = new Node();
        tmpNode.setLabel(father + "." + k);
        tmpNode.setName(this.computeNodeName(term));
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpTransition = new Transition();
        tmpTransition.setSource(father + "." + (k - 1));
        tmpTransition.setTarget(father + "." + k);
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        k = k + 1;
      }
      tmpNode = new Node();
      tmpNode.setLabel(father + "." + k);
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_END);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(father + "." + (k - 1));
      tmpTransition.setTarget(father + "." + k);
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
    } else if (orMatcher.find()
        && (orMatcher.group().length() == expression.length())) {
      expressionType = 2;
      System.out
          .print("an or expression processed: " + expression + "\n");
      nodes.get(father).setType(Node.TYPE_OR);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".1");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_START);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".2");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_CONDITION);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(father + ".1");
      tmpTransition.setTarget(father + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      int k = 3;
      for (Iterator<String> iterator = this.findTermsInExpression(
          expression, "|").iterator(); iterator.hasNext();) {
        String term = iterator.next();
```

```
        tmpNode = new Node();
        tmpNode.setLabel(father + "." + k);
        tmpNode.setName(this.computeNodeName(term));
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpTransition = new Transition();
        tmpTransition.setSource(father + ".2");
        tmpTransition.setTarget(father + "." + k);
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        k = k + 1;
      }
      tmpNode = new Node();
      tmpNode.setLabel(father + "." + k);
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_END);
      nodes.put(tmpNode.getLabel(), tmpNode);
      int tmpCounter = k;
      for (k = tmpCounter - 1; k > 2; k--) {
        tmpTransition = new Transition();
        tmpTransition.setSource(father + "." + k);
        tmpTransition.setTarget(father + "." + tmpCounter);
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
      }
    } else if (parallelMatcher.find()
        && (parallelMatcher.group().length() == expression.length())) {
      expressionType = 1;
      System.out.print("a parallel expression processed: " + expression
          + "\n");
      nodes.get(father).setType(Node.TYPE_OR);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".1");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_START);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".2");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_AND);
      // tmpNode.setName(expression);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(father + ".1");
      tmpTransition.setTarget(father + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".3");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_END);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(father + ".2");
      tmpTransition.setTarget(father + ".3");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      int k = 1;
      for (Iterator<String> iterator = this.findTermsInExpression(
          expression, "||").iterator(); iterator.hasNext();) {
        String term = iterator.next();
        tmpNode = new Node();
        tmpNode.setType(Node.TYPE_OR);
        tmpNode.setLabel(father + ".2." + k);
        tmpNode.setName(tmpNode.getLabel());
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpNode = new Node();
        tmpNode.setLabel(father + ".2." + k + ".1");
        tmpNode.setName(tmpNode.getLabel());
```

```
      tmpNode.setType(Node.TYPE_START);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".2." + k + ".2");
      tmpNode.setName(this.computeNodeName(term));
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(father + ".2." + k + ".1");
      tmpTransition.setTarget(father + ".2." + k + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      tmpNode = new Node();
      tmpNode.setLabel(father + ".2." + k + ".3");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_END);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(father + ".2." + k + ".2");
      tmpTransition.setTarget(father + ".2." + k + ".3");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      k = k + 1;
    }
  }
  List<String> myTerms = null;
  switch (expressionType) {
  case 0:
    myTerms = new LinkedList<String>();
    myTerms.add(expression);
    break;
  case 1:
    myTerms = this.findTermsInExpression(expression, "||");
    break;
  case 2:
    myTerms = this.findTermsInExpression(expression, "|");
    break;
  case 3:
    myTerms = this.findTermsInExpression(expression, ".");
    break;
  }
  for (Iterator<String> iterator = myTerms.iterator(); iterator.hasNext();) {
    String term = iterator.next();
    // pattern for basicTerm
    Pattern patternBasicTerm = Pattern.compile("^\\w+$");
    Matcher basicTermMatcher = patternBasicTerm.matcher(term);
    // pattern for (term)
    Pattern patternComplexParenthesisTerm = Pattern
        .compile("^\\(.+\\)$");
    Matcher complexParenthesisTermMatcher = patternComplexParenthesisTerm
        .matcher(term);
    // pattern for [term]
    Pattern patternComplexOptionalTerm = Pattern.compile("^\\[.+\\]$");
    Matcher complexOptionalTermMatcher = patternComplexOptionalTerm
        .matcher(term);
    // pattern for termω
    Pattern patternForeverTerm = Pattern.compile(".+ω$");
    Matcher foreverTermMatcher = patternForeverTerm.matcher(term);
    // pattern for term+
    Pattern patternOneOrMoreTimesTerm = Pattern.compile(".+\\+$");
    Matcher oneOrMoreTimesTermMatcher = patternOneOrMoreTimesTerm
        .matcher(term);
    // pattern for term*
    Pattern patternZeroOrMoreTimesTerm = Pattern.compile(".+\\*$");
    Matcher zeroOrMoreTimesTermMatcher = patternZeroOrMoreTimesTerm
        .matcher(term);
    // pattern for |termω|n
    Pattern patternParallelManyTimesTerm = Pattern
        .compile("^\\|.+(ω\\|(\\d)+)$");
    Matcher parallelManyTimesTermMatcher = patternParallelManyTimesTerm
        .matcher(term);
    if (complexParenthesisTermMatcher.find()
        && (complexParenthesisTermMatcher.group().length() == term
```

```
            .length())) {
        this.createStatechart(term.substring(1, term.length() - 1),
            this.getNode(father, term));
      } else if (complexOptionalTermMatcher.find()
          && (complexOptionalTermMatcher.group().length() == term
              .length())) {
        this.nodes.get(getNode(father, term)).setType(
            Node.TYPE_OR);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".1");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_START);
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".2");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_CONDITION);
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".3");
        String insideTerm = term.substring(1, term.length() - 1);
        tmpNode.setName(this.computeNodeName(insideTerm));
        nodes.put(tmpNode.getLabel(), tmpNode);
        basicTermMatcher = patternBasicTerm.matcher(insideTerm);
        if (basicTermMatcher.find()
            && (basicTermMatcher.group().length() == insideTerm
                .length())) {
          this.handleBasicTerm(insideTerm, this.getNode(father, term)
              + ".3");
        } else {
          this.createStatechart(insideTerm, this
              .getNode(father, term)
              + ".3");
        }
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".4");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_END);
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".1");
        tmpTransition.setTarget(this.getNode(father, term) + ".2");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".2");
        tmpTransition.setTarget(this.getNode(father, term) + ".3");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".2");
        tmpTransition.setTarget(this.getNode(father, term) + ".4");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".3");
        tmpTransition.setTarget(this.getNode(father, term) + ".4");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
      } else if (zeroOrMoreTimesTermMatcher.find()
          && (zeroOrMoreTimesTermMatcher.group().length() == term
              .length())) {
        this.nodes.get(getNode(father, term)).setType(
            Node.TYPE_OR);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".1");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_START);
```

```
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".2");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_CONDITION);
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".3");
        String insideTerm = term.substring(0, term.length() - 1);
        tmpNode.setName(this.computeNodeName(insideTerm));
        nodes.put(tmpNode.getLabel(), tmpNode);
        basicTermMatcher = patternBasicTerm.matcher(insideTerm);
        if (basicTermMatcher.find()
            && (basicTermMatcher.group().length() == insideTerm
                .length())) {
          this.handleBasicTerm(insideTerm, this.getNode(father, term)
              + ".3");
        } else {
          this.createStatechart(insideTerm, this
              .getNode(father, term)
              + ".3");
        }
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".4");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_END);
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".1");
        tmpTransition.setTarget(this.getNode(father, term) + ".2");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".2");
        tmpTransition.setTarget(this.getNode(father, term) + ".3");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".2");
        tmpTransition.setTarget(this.getNode(father, term) + ".4");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".3");
        tmpTransition.setTarget(this.getNode(father, term) + ".3");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
        tmpTransition = new Transition();
        tmpTransition.setSource(this.getNode(father, term) + ".3");
        tmpTransition.setTarget(this.getNode(father, term) + ".4");
        tmpTransition.setName(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget());
        transitions.put(tmpTransition.getSource() + "TO"
            + tmpTransition.getTarget(), tmpTransition);
      } else if (foreverTermMatcher.find()
          && (foreverTermMatcher.group().length() == term.length())) {
        this.nodes.get(getNode(father, term)).setType(Node.TYPE_OR);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".1");
        tmpNode.setName(tmpNode.getLabel());
        tmpNode.setType(Node.TYPE_START);
        nodes.put(tmpNode.getLabel(), tmpNode);
        tmpNode = new Node();
        tmpNode.setLabel(getNode(father, term) + ".2");
        String insideTerm = term.substring(0, term.length() - 1);
        tmpNode.setName(this.computeNodeName(insideTerm));
        nodes.put(tmpNode.getLabel(), tmpNode);
        basicTermMatcher = patternBasicTerm.matcher(insideTerm);
```

```
      if (basicTermMatcher.find()
          && (basicTermMatcher.group().length() == insideTerm
              .length())) {
        this.handleBasicTerm(insideTerm, this.getNode(father, term)
            + ".2");
      } else {
        this.createStatechart(insideTerm, this
            .getNode(father, term)
            + ".2");
      }
      tmpTransition = new Transition();
      tmpTransition.setSource(this.getNode(father, term) + ".1");
      tmpTransition.setTarget(this.getNode(father, term) + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      tmpTransition = new Transition();
      tmpTransition.setSource(this.getNode(father, term) + ".2");
      tmpTransition.setTarget(this.getNode(father, term) + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
    } else if (oneOrMoreTimesTermMatcher.find()
        && (oneOrMoreTimesTermMatcher.group().length() == term
            .length())) {
      this.nodes.get(getNode(father, term)).setType(Node.TYPE_OR);
      tmpNode = new Node();
      tmpNode.setLabel(getNode(father, term) + ".1");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_START);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpNode = new Node();
      String insideTerm = term.substring(0, term.length() - 1);
      tmpNode.setLabel(getNode(father, term) + ".2");
      tmpNode.setName(this.computeNodeName(insideTerm));
      nodes.put(tmpNode.getLabel(), tmpNode);
      basicTermMatcher = patternBasicTerm.matcher(insideTerm);
      if (basicTermMatcher.find()
          && (basicTermMatcher.group().length() == insideTerm
              .length())) {
        this.handleBasicTerm(insideTerm, this.getNode(father, term)
            + ".2");
      } else {
        this.createStatechart(insideTerm, this
            .getNode(father, term)
            + ".2");
      }
      tmpNode = new Node();
      tmpNode.setLabel(getNode(father, term) + ".3");
      tmpNode.setName(tmpNode.getLabel());
      tmpNode.setType(Node.TYPE_END);
      nodes.put(tmpNode.getLabel(), tmpNode);
      tmpTransition = new Transition();
      tmpTransition.setSource(this.getNode(father, term) + ".1");
      tmpTransition.setTarget(this.getNode(father, term) + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      tmpTransition = new Transition();
      tmpTransition.setSource(this.getNode(father, term) + ".2");
      tmpTransition.setTarget(this.getNode(father, term) + ".2");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
      tmpTransition = new Transition();
      tmpTransition.setSource(this.getNode(father, term) + ".2");
      tmpTransition.setTarget(this.getNode(father, term) + ".3");
      tmpTransition.setName(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget());
      transitions.put(tmpTransition.getSource() + "TO"
          + tmpTransition.getTarget(), tmpTransition);
    } else if (parallelManyTimesTermMatcher.find()
```

```
          && (parallelManyTimesTermMatcher.group().length() == term
              .length())) {
        this.nodes.get(getNode(father, term)).setType(Node.TYPE_AND);
        for (int k = 1;k<=Integer.parseInt(term.substring(term.lastIndexOf("|")+1,
term.length())); k++) {
          tmpNode = new Node();
          tmpNode.setType(Node.TYPE_OR);
          tmpNode.setLabel(getNode(father, term) + "." + k);
          tmpNode.setName(tmpNode.getLabel());
          nodes.put(tmpNode.getLabel(), tmpNode);
          tmpNode = new Node();
          tmpNode.setLabel(getNode(father, term) + "." + k + ".1");
          tmpNode.setName(tmpNode.getLabel());
          tmpNode.setType(Node.TYPE_START);
          nodes.put(tmpNode.getLabel(), tmpNode);
          tmpNode = new Node();
          tmpNode.setLabel(getNode(father, term) + "." + k + ".2");
          String insideTerm = term.substring(term.indexOf("|")+1,
term.lastIndexOf("ω"));
          tmpNode.setName(this.computeNodeName(insideTerm));
          nodes.put(tmpNode.getLabel(), tmpNode);
          basicTermMatcher = patternBasicTerm.matcher(insideTerm);
          if (basicTermMatcher.find()
              && (basicTermMatcher.group().length() == insideTerm
                  .length())) {
            this.handleBasicTerm(insideTerm, getNode(father, term) + "." + k + ".2");
          } else {
            this.createStatechart(insideTerm, getNode(father, term) + "." + k + ".2");
          }
          tmpNode = new Node();
          tmpNode.setLabel(getNode(father, term) + "." + k + ".3");
          tmpNode.setName(tmpNode.getLabel());
          tmpNode.setType(Node.TYPE_END);
          nodes.put(tmpNode.getLabel(), tmpNode);
          tmpTransition = new Transition();
          tmpTransition.setSource(getNode(father, term) + "." + k + ".1");
          tmpTransition.setTarget(getNode(father, term) + "." + k + ".2");
          tmpTransition.setName(tmpTransition.getSource() + "TO"
              + tmpTransition.getTarget());
          transitions.put(tmpTransition.getSource() + "TO"
              + tmpTransition.getTarget(), tmpTransition);
          tmpTransition = new Transition();
          tmpTransition.setSource(getNode(father, term) + "." + k + ".2");
          tmpTransition.setTarget(getNode(father, term) + "." + k + ".3");
          tmpTransition.setName(tmpTransition.getSource() + "TO"
              + tmpTransition.getTarget());
          transitions.put(tmpTransition.getSource() + "TO"
              + tmpTransition.getTarget(), tmpTransition);
        }
      } else if (basicTermMatcher.find()
          && (basicTermMatcher.group().length() == term.length())) {
        this.handleBasicTerm(term, this.getNode(father, term));
      }

    }
  }

  public String parent(String node) {
    return (node.substring(0, node.lastIndexOf(".")));
  }

  public void handleBasicTerm(String term, String node) {
    boolean isBasic = true;
    if (formulas.containsKey(term)) {
      this.nodes.get(node).setType(Node.TYPE_OR);
      this.createStatechart(formulas.get(term), node);
      isBasic = false;
    }
    if (isBasic) {
      nodes.get(node).setType(Node.TYPE_BASIC);
    }
  }

  public String computeNodeName(String term) {
    String nodeName = new String(term);
    nodeName = nodeName.replaceAll("\\|\\|", "_parallel_");
```

```
    nodeName = nodeName.replaceAll("ω", "_forever_");
    nodeName = nodeName.replaceAll("\\.", "_sequence_");
    nodeName = nodeName.replaceAll("\\|", "_or_");
    nodeName = nodeName.replaceAll("\\*", "_zero_or_more_times_");
    nodeName = nodeName.replaceAll("\\+", "_one_or_more_times_");
    nodeName = nodeName.replaceAll("\\(", "_open_group_");
    nodeName = nodeName.replaceAll("\\)", "_close_group_");
    nodeName = nodeName.replaceAll("\\[", "_open_option_");
    nodeName = nodeName.replaceAll("\\]", "_close_option_");
    return nodeName;
  }

  public String getNode(String father, String term) {
    LinkedList<String> queue = new LinkedList<String>();
    queue.addLast(father);
    while (queue.size() > 0) {
      String first = queue.removeFirst();
      if (nodes.get(first).getName().equalsIgnoreCase(
          this.computeNodeName(term))) {
        return first;
      } else {
        for (Iterator<String> iterator = getSons(first).iterator(); iterator
            .hasNext();) {
          String son = iterator.next();
          queue.addLast(son);
        }
      }
    }
    return null;
  }

  public List<String> getSons(String father) {
    int i = 1;
    List<String> results = new LinkedList<String>();
    while (nodes.containsKey(father + "." + i)) {
      results.add(father + "." + i);
      i++;
    }
    return results;
  }

  public List<String> findTermsInExpression(String expression,
      String connector) {
    List<String> foundTerms = new LinkedList<String>();
    StringTokenizer t = new StringTokenizer(expression, connector);
    String currentTerm = new String();
    while (t.hasMoreTokens()) {
      int parenthesis = 0;
      currentTerm = currentTerm + t.nextToken();
      for (int i = 0; i < currentTerm.length(); i++) {
        if (currentTerm.regionMatches(i, "(", 0, 1))
          parenthesis++;
        if (currentTerm.regionMatches(i, ")", 0, 1))
          parenthesis--;
      }
      if (parenthesis == 0) {
        foundTerms.add(currentTerm);
        System.out.print("found term: " + currentTerm + "\n");
        currentTerm = new String();
      } else
        currentTerm = currentTerm + connector;
    }
    return foundTerms;
  }
}
```

Listing 25. The Model.java file

```
package fr.parisdescartes.mi.aseme.t2m.srm2iac;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
```

```
public class Model {
  List<String> nodes = new LinkedList<String>();
  List<String> transitions = new LinkedList<String>();
  String name = null;

  public List<String> getNodes() {
    return nodes;
  }
  public void setNodes(List<String> nodes) {
    this.nodes = nodes;
  }
  public void addNode(String newNode) {
    this.nodes.add(newNode);
  }
  public List<String> getTransitions() {
    return transitions;
  }
  public void setTransitions(List<String> transitions) {
    this.transitions = transitions;
  }
  public void addTransition(String newTransition) {
    this.transitions.add(newTransition);
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String toHutnString() {
    String modelType = "Model";
    String result = new String(" " + modelType + " \""
        + this.getName() + "\"{\n" + "  name: \""
        + this.getName() + "\"\n");
    if (this.nodes.size() > 0) {
      result = result + "  nodes: ";
      boolean first = true;
      for (Iterator<String> iterator = nodes.iterator(); iterator
          .hasNext();) {
        if (!first)
          result = result + ", ";
        result = result + "Node \"" + iterator.next() + "\"";
        first = false;
      }
      result = result + "\n";
    }
    if (this.transitions.size() > 0) {
      result = result + "  transitions: ";
      boolean first = true;
      for (Iterator<String> iterator = transitions.iterator(); iterator
          .hasNext();) {
        if (!first)
          result = result + ", ";
        result = result + "Transition \"" + iterator.next() + "\"";
        first = false;
      }
      result = result + "\n";
    }
    result = result + " }\n";
    return result;
  }
}
```

Listing 26. The Node.java file

```
package fr.parisdescartes.mi.aseme.t2m.srm2iac;

import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class Node {
  public static final String TYPE_OR = "OR";
```

```
  public static final String TYPE_AND = "AND";
  public static final String TYPE_START = "START";
  public static final String TYPE_END = "END";
  public static final String TYPE_CONDITION = "CONDITION";
  public static final String TYPE_BASIC = "BASIC";
  String type = null;
  String name = null;
  String label = null;
  String activity = null;
  List<String> variables = new LinkedList<String>();

  public List<String> getVariables() {
    return variables;
  }
  public void setVariables(List<String> variables) {
    this.variables = variables;
  }
  public void addVariable(String newVariable) {
    this.variables.add(newVariable);
  }
  public String getType() {
    return type;
  }
  public void setType(String type) {
    this.type = type;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getLabel() {
    return label;
  }
  public void setLabel(String label) {
    this.label = label;
  }
  public String getActivity() {
    return activity;
  }
  public void setActivity(String activity) {
    this.activity = activity;
  }
  public String toHutnString() {
    String nodeType = "Node";//(this.getLabel().equalsIgnoreCase("0"))?"root":"Node";
    String result = new String(" "+nodeType+" \"" + this.getLabel() + "\"{\n"
        + "  type: \"" + this.getType() + "\"\n" + "  name: \""
        + this.getName() + "\"\n" + "  label: \"" + this.getLabel()
        + "\"\n" + "  activity: \"" + this.getActivity() + "\"\n");
    if (this.variables.size() > 0) {
      result = result + "  variables: ";
      boolean first = true;
      for (Iterator<String> iterator = variables.iterator(); iterator
          .hasNext();) {
        if (!first)
          result = result + ", ";
        result = result + "Variable \"" + iterator.next() + "\"";
        first = false;
      }
      result = result + "\n";
    }
    result = result + " }\n";
    return result;
  }
}
```

Listing 27. The Transition.java file

```
package fr.parisdescartes.mi.aseme.t2m.srm2iac;

public class Transition {
  String source = null;
  String target = null;
```

```
  String TE = null;
  String name = null;
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String getSource() {
    return source;
  }
  public void setSource(String source) {
    this.source = source;
  }
  public String getTarget() {
    return target;
  }
  public void setTarget(String target) {
    this.target = target;
  }
  public String getTE() {
    return TE;
  }
  public void setTE(String te) {
    TE = te;
  }
  public String toHutnString(){
    return new String(" Transition \""+this.getName()+"\"{\n" +
        "  name: \""+this.getName()+"\"\n" +
        "  TE: \""+this.getTE()+"\"\n" +
        "  source: Node \""+this.getSource()+"\"\n" +
        "  target: Node \""+this.getTarget()+"\"\n" +
        " }\n");
  }
}
```

Listing 28. The Variable.java file

```
package fr.parisdescartes.mi.aseme.t2m.srm2iac;

public class Variable {
  String type=null;
  String name =null;
  public String getType() {
    return type;
  }
  public void setType(String type) {
    this.type = type;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
  public String toHutnString(){
    return new String(" Variable \""+this.getName()+"\"{\n" +
        "  type: \""+this.getType()+"\"\n" +
        "  name: \""+this.getName()+"\"\n" +
        " }\n");
  }
}
```

Annex 5.

The IAC2JADE transformation project files

Listing 29. The agent xpand template file (Agent.xpt)

```
«IMPORT IAC»
«EXTENSION fr::parisdescartes::mi::aseme::m2t::IACmodel::nodeHelper»
«EXTENSION fr::parisdescartes::mi::aseme::m2t::IACmodel::ComplexBehaviourHelper»

«DEFINE javaClass FOR IAC::Model»
  «LET name AS packageName»
    «EXPAND nodeClass(packageName, this) FOREACH nodes»
    «EXPAND variableHolderClass(packageName, this) FOREACH variables»
  «ENDLET»
«ENDDEFINE»

«DEFINE variableHolderClass(String packageName, Model model) FOR IAC::Variable»
  «FILE variableHolderFileName()»
  package «packageName»;
  import jade.core.behaviours.Behaviour;
  «IF type.compareTo("ACLMessage")==0»import jade.lang.acl.ACLMessage;«ENDIF»

  public class «type»Holder {
    «type» «lowerCaseFirstCharacterOfVariable(this)» = null;
    Behaviour owner;

    public «type»Holder(Behaviour owner) {
      super();
      this.owner = owner;
    }

    public «type» get«type»() {
      return «lowerCaseFirstCharacterOfVariable(this)»;
    }

    public void set«type»(«type» «lowerCaseFirstCharacterOfVariable(this)») {
      this.«lowerCaseFirstCharacterOfVariable(this)» =
«lowerCaseFirstCharacterOfVariable(this)»;
    }

    public Behaviour getOwner() {
```

```
      return owner;
    }
  }
  «ENDFILE»
«ENDDEFINE»

«DEFINE nodeClass(String packageName, Model model) FOR IAC::Node»
  «IF !name.startsWith("0")»
    «IF label.compareTo("0")==0»
      «EXPAND agentClass(packageName, model) FOR this»
    «ELSE»
      «EXPAND behaviourClass(packageName, model) FOR this»
    «ENDIF»
  «ENDIF»
«ENDDEFINE»

«DEFINE agentClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
     package «packageName»;

    import jade.core.Agent;

       public class «className()» extends Agent{

      public void setup(){
        //add behaviour
        addBehaviour(«getAgentBehaviour(this,model)»);
      }
      protected void takeDown() {
      doDelete();
    }
  }
   «ENDFILE»
«ENDDEFINE»

«DEFINE behaviourClass(String packageName, Model model) FOR IAC::Node»
  «IF type.compareTo("BASIC")==0»
    «EXPAND simpleBehaviourClass(packageName, model) FOR this»
  «ELSE»
    «IF determineBehaviourType(this,model).compareTo("parallel")==0»
      «EXPAND parallelBehaviourClass(packageName, model) FOR this»
    «ELSEIF (determineBehaviourType(this,model).compareTo("sequence")==0) ||
(determineBehaviourType(this,model).compareTo("or")==0)»
      «EXPAND sequenceBehaviourClass(packageName, model) FOR this»
    «ELSEIF determineBehaviourType(this,model).compareTo("forever")==0»
      «EXPAND cyclicBehaviourClass(packageName, model) FOR this»
    «ELSEIF determineBehaviourType(this,model).compareTo("one_or_more_times")==0»
      «EXPAND simpleOneOrMoreBehaviourClass(packageName, model) FOR this»
    «ELSEIF determineBehaviourType(this,model).compareTo("zero_or_more_times")==0»
      «EXPAND simpleZeroOrMoreBehaviourClass(packageName, model) FOR this»
    «ENDIF»
  «ENDIF»
«ENDDEFINE»

«DEFINE simpleZeroOrMoreBehaviourClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
     package «packageName»;

    import jade.core.Agent;
    import jade.core.behaviours.Behaviour;
    import jade.core.behaviours.SimpleBehaviour;

    public class «className()» extends SimpleBehaviour{

          «addVariables(this,model)»
          protected boolean finished;
          Behaviour simpleZeroOrMoreTimesBehaviour = null;

            public «className()»(Agent a«getParams(this,model)»){
        super(a);
        «instantiateParams(this,model)»
            if («getTransitionToChildOf(this,model)»){
          simpleZeroOrMoreTimesBehaviour = new «getCyclicBehaviour(this,model)»;
          myAgent.addBehaviour(simpleZeroOrMoreTimesBehaviour);
          finished = false;
        }
```

```
        else finished = true;
      }

          public void action(){
           if (simpleZeroOrMoreTimesBehaviour.done()){
             if («getTransitionToSelfConditionOfChild(this,model)»){
            simpleZeroOrMoreTimesBehaviour = new «getCyclicBehaviour(this,model)»;
            myAgent.addBehaviour(simpleZeroOrMoreTimesBehaviour);
          }
          else finished = true;
        }
           }

           public boolean done() {
          return finished;
        }

      }
     «ENDFILE»
«ENDDEFINE»

«DEFINE simpleOneOrMoreBehaviourClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
     package «packageName»;

    import jade.core.Agent;
    import jade.core.behaviours.Behaviour;
    import jade.core.behaviours.SimpleBehaviour;

    public class «className()» extends SimpleBehaviour{

          «addVariables(this,model)»
          protected boolean finished;
          Behaviour simpleOneOrMoreTimesBehaviour = null;

            public «className()»(Agent a«getParams(this,model)»){
        super(a);
        «instantiateParams(this,model)»
        finished = false;
        simpleOneOrMoreTimesBehaviour = new «getCyclicBehaviour(this,model)»;
        myAgent.addBehaviour(simpleOneOrMoreTimesBehaviour);
           }

           public void action(){
            if (simpleOneOrMoreTimesBehaviour.done()){
              if («getTransitionToSelfConditionOfChild(this,model)»){
            simpleOneOrMoreTimesBehaviour = new «getCyclicBehaviour(this,model)»;
            myAgent.addBehaviour(simpleOneOrMoreTimesBehaviour);
          }
          else finished = true;
        }
           }

           public boolean done() {
          return finished;
        }

      }
     «ENDFILE»
«ENDDEFINE»

«DEFINE cyclicBehaviourClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
    package «packageName»;

  import jade.core.Agent;
  import jade.core.behaviours.Behaviour;
  import jade.core.behaviours.CyclicBehaviour;

  public class «className()» extends CyclicBehaviour{

      «addVariables(this,model)»
      Behaviour foreverBehaviour = null;

    public «className()»(Agent a«getParams(this, model)») {
      super(a);
```

```
      «instantiateParams(this,model)»
      foreverBehaviour = new «getCyclicBehaviour(this,model)»;
      myAgent.addBehaviour(foreverBehaviour);
    }
    public void action() {
      if (foreverBehaviour.done()){
        foreverBehaviour = new «getCyclicBehaviour(this,model)»;
        myAgent.addBehaviour(foreverBehaviour);
      }
    }
  }
   «ENDFILE»
«ENDDEFINE»

«DEFINE sequenceBehaviourClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
    package «packageName»;

  import jade.core.Agent;
  import jade.core.behaviours.SequentialBehaviour;

  public class «className()» extends SequentialBehaviour {

      «addVariables(this,model)»

    public «className()»(Agent a«getParams(this, model)») {
      super(a);
      «instantiateParams(this,model)»
      «IF determineBehaviourType(this,model).compareTo("or")==0»
        «addConditionalSubBehaviour(this,model)»
      «ELSE»
        «addSubBehaviours(this,model)»
      «ENDIF»
    }
  }
   «ENDFILE»
«ENDDEFINE»

«DEFINE parallelBehaviourClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
    package «packageName»;

  import jade.core.Agent;
  import jade.core.behaviours.ParallelBehaviour;
  import jade.core.behaviours.ThreadedBehaviourFactory;

  public class «className()» extends ParallelBehaviour{

    ThreadedBehaviourFactory tbf = null;
      «addVariables(this,model)»

        public «className()»(Agent a«getParams(this, model)»){
      super(a,ParallelBehaviour.WHEN_ALL);
      «instantiateParams(this,model)»
      tbf = new ThreadedBehaviourFactory();
      «addParallelBehaviours(this,model)»
        }
    }
   «ENDFILE»
«ENDDEFINE»

«DEFINE simpleBehaviourClass(String packageName, Model model) FOR IAC::Node»
   «FILE classFileName()»
     package «packageName»;

    import jade.core.Agent;
    import jade.core.behaviours.SimpleBehaviour;
    «importMessageClasses(this)»

    public class «className()» extends SimpleBehaviour{

        «addVariables(this,model)»
      «addMessageTemplateVariable(this)»
      boolean finished = false;

          public «className()»(Agent a«getParams(this,model)»){
```

```
        super(a);
        «instantiateParams(this,model)»
            }

           public void action(){
              «addAction(this, model)»
           }

           public boolean done() {
          return finished;
        }

      }
     «ENDFILE»
«ENDDEFINE»
```

Listing 30. The *nodeHelper* xtend file (NodeHelper.ext)

```
import IAC;

String className( Node e ) :
   JAVA fr.parisdescartes.mi.aseme.m2t.IACmodel.NodeHelper.className(IAC.Node);

String classFileName( Node e ) :
   JAVA fr.parisdescartes.mi.aseme.m2t.IACmodel.NodeHelper.classFileName(IAC.Node);

String variableFileName( Variable e ) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.NodeHelper.variableFileName(IAC.Variable);

String variableHolderFileName( Variable e ) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.NodeHelper.variableHolderFileName(IAC.Variable
);

String lowerCaseFirstCharacterOfVariable( Variable e ) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.NodeHelper.lowerCaseFirstCharacterOfVariable(I
AC.Variable);
```

Listing 31. The nodeHelper Java implementation class (NodeHelper.java)

```
package fr.parisdescartes.mi.aseme.m2t.IACmodel;

import IAC.Node;
import IAC.Variable;

public class NodeHelper {

  public static String className( Node e ) {
    return
((e.getLabel().equalsIgnoreCase("0"))?e.getName()+"Agent":e.getName()+"Behaviour");
  }

  public static String classFileName( Node e ) {
    return className(e)+".java";
  }

  public static String variableFileName( Variable e ) {
      return e.getType()+".java";
    }

  public static String variableHolderFileName( Variable e ) {
      return e.getType()+"Holder.java";
    }

  public static String lowerCaseFirstCharacterOfVariable( Variable e ) {
      return new String(e.getType().substring(0,
1).toLowerCase()+e.getType().substring(1));
    }
}
```

Listing 32. The *ComplexBehaviourHelper* xtend file (ComplexBehaviourHelper.ext)

```
import IAC;

String addVariables( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.addVariables(IAC.Node,
IAC.Model);

String getParams( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.getParams(IAC.Node,
IAC.Model);

String getCyclicBehaviour( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.getCyclicBehaviour(IAC.
Node, IAC.Model);

String getAgentBehaviour( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.getAgentBehaviour(IAC.N
ode, IAC.Model);

String getTransitionToSelfConditionOfChild( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.getTransitionToSelfCond
itionOfChild(IAC.Node, IAC.Model);

String addAction( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.addAction(IAC.Node,
IAC.Model);

String addMessageTemplateVariable( Node e ) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.addMessageTemplateVaria
ble(IAC.Node);

String importMessageClasses( Node e ) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.importMessageClasses(IA
C.Node);

String getTransitionToChildOf( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.getTransitionToChildOf(
IAC.Node, IAC.Model);

String instantiateParams( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.instantiateParams(IAC.N
ode, IAC.Model);

String addParallelBehaviours( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.addParallelBehaviours(I
AC.Node, IAC.Model);

String addSubBehaviours( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.addSubBehaviours(IAC.No
de, IAC.Model);

String addConditionalSubBehaviour( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.addConditionalSubBehavi
our(IAC.Node, IAC.Model);

List subBehavioursOf( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.subBehavioursOf(IAC.Nod
e, IAC.Model);
```

```
String determineBehaviourType( Node e , Model m) :
   JAVA
fr.parisdescartes.mi.aseme.m2t.IACmodel.ComplexBehaviourHelper.determineBehaviourType(
IAC.Node, IAC.Model);
```

Listing 33. The ComplexBehaviourHelper Java implementation class (ComplexBehaviourHelper.java)

```
package fr.parisdescartes.mi.aseme.m2t.IACmodel;

import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.eclipse.emf.common.util.BasicEList;
import org.eclipse.emf.common.util.EList;

import IAC.Model;
import IAC.Node;
import IAC.Transition;
import IAC.Variable;

public class ComplexBehaviourHelper {

  public static final String TYPE_FOREVER = "forever";
  public static final String TYPE_ONE_OR_MORE_TIMES = "one_or_more_times";
  public static final String TYPE_ZERO_OR_MORE_TIMES = "zero_or_more_times";
  public static final String TYPE_SEQUENCE = "sequence";
  public static final String TYPE_OR = "or";
  public static final String TYPE_PARALLEL = "parallel";

  public static boolean existConditionInSubNodesOf(Node e, Model m) {
    boolean result = false;
    // TO DO define the attributes
    EList<Node> tmp = subNodesOf(e, m);
    for (Iterator<Node> iterator = tmp.iterator(); iterator.hasNext();) {
      Node node = iterator.next();
      if (node.getType().equalsIgnoreCase("CONDITION")) {
        result = true;
      }
    }
    return result;
  }

  public static String addParallelBehaviours(Node e, Model m) {
    String result = new String();
    for (Iterator<Node> iterator = subBehavioursOf(e, m).iterator(); iterator
        .hasNext();) {
      Node node = iterator.next();
      String params = new String();
      boolean firstParam = true;
      for (Iterator<Variable> iterator2 = node.getVariables().iterator(); iterator2
          .hasNext();) {
        Variable tmpVar2 = iterator2.next();
        for (Iterator<Variable> iterator3 = e.getVariables().iterator(); iterator3
            .hasNext();) {
          Variable tmpVar3 = iterator3.next();
          if (tmpVar2.getName().equalsIgnoreCase(tmpVar3.getName())) {
            params = params + ", " + tmpVar2.getName();
            if (firstParam)
              firstParam = false;
          }
        }
      }
      result = result + "\nmyAgent.addBehaviour(tbf.wrap(new "
          + node.getName() + "Behaviour(this.myAgent" + params
          + ")));";
    }
    return result;
  }
```

```
public static String getCyclicBehaviour(Node e, Model m) {
  String result = new String();
  for (Iterator<Node> iterator = subBehavioursOf(e, m).iterator(); iterator
      .hasNext();) {
    Node node = iterator.next();
    String params = new String();
    boolean firstParam = true;
    for (Iterator<Variable> iterator2 = node.getVariables().iterator(); iterator2
        .hasNext();) {
      Variable tmpVar2 = iterator2.next();
      for (Iterator<Variable> iterator3 = e.getVariables().iterator(); iterator3
          .hasNext();) {
        Variable tmpVar3 = iterator3.next();
        if (tmpVar2.getName().equalsIgnoreCase(tmpVar3.getName())) {
          params = params + ", " + tmpVar2.getName();
          if (firstParam)
            firstParam = false;
        }
      }
    }
    result = result + node.getName() + "Behaviour(this.myAgent"
        + params + ")";
    break;
  }
  return result;
}

public static String getAgentBehaviour(Node e, Model m) {
  String result = new String();
  for (Iterator<Node> iterator = subBehavioursOf(e, m).iterator(); iterator
      .hasNext();) {
    Node node = iterator.next();
    String params = new String();
    boolean firstParam = true;
    for (Iterator<Variable> iterator2 = node.getVariables().iterator(); iterator2
        .hasNext();) {
      Variable tmpVar2 = iterator2.next();
      for (Iterator<Variable> iterator3 = e.getVariables().iterator(); iterator3
          .hasNext();) {
        Variable tmpVar3 = iterator3.next();
        if (tmpVar2.getName().equalsIgnoreCase(tmpVar3.getName())) {
          params = params + ", " + tmpVar2.getName();
          if (firstParam)
            firstParam = false;
        }
      }
    }
    result = result + "new " + node.getName() + "Behaviour(this"
        + params + ")";
    break;
  }
  return result;
}

public static String addSubBehaviours(Node e, Model m) {
  String result = new String();
  for (Iterator<Node> iterator = sortSubNodes(subBehavioursOf(e, m))
      .iterator(); iterator.hasNext();) {
    Node node = iterator.next();
    String params = new String();
    boolean firstParam = true;
    for (Iterator<Variable> iterator2 = node.getVariables().iterator(); iterator2
        .hasNext();) {
      Variable tmpVar2 = iterator2.next();
      for (Iterator<Variable> iterator3 = e.getVariables().iterator(); iterator3
          .hasNext();) {
        Variable tmpVar3 = iterator3.next();
        if (tmpVar2.getName().equalsIgnoreCase(tmpVar3.getName())) {
          params = params + ", " + tmpVar2.getName();
          if (firstParam)
            firstParam = false;
        }
      }
    }
    result = result + "\naddSubBehaviour(new " + node.getName()
        + "Behaviour(this.myAgent" + params + "));";
```

```
    }
    return result;
  }

  public static String addConditionalSubBehaviour(Node e, Model m) {
    String result = new String();
    EList<Node> tmp = subBehavioursOf(e, m);
    boolean allowedNullOnce = false;
    for (int i = 0; i < tmp.size(); i++) {
      Node node = tmp.get(i);
      String params = new String();
      boolean firstParam = true;
      for (Iterator<Variable> iterator2 = node.getVariables().iterator(); iterator2
          .hasNext();) {
        Variable tmpVar2 = iterator2.next();
        for (Iterator<Variable> iterator3 = e.getVariables().iterator(); iterator3
            .hasNext();) {
          Variable tmpVar3 = iterator3.next();
          if (tmpVar2.getName().equalsIgnoreCase(tmpVar3.getName())) {
            params = params + ", " + tmpVar2.getName();
            if (firstParam)
              firstParam = false;
          }
        }
      }
      String condition = null;
      for (Iterator<Transition> iterator = m.getTransitions().iterator(); iterator
          .hasNext();) {
        Transition tmpT = iterator.next();
        if (tmpT.getTarget().getLabel().equalsIgnoreCase(
            node.getLabel())) {
          condition = getConditionOfExpression(tmpT.getTE());
        }
      }
      if (condition == null)
        condition = "null";
      if ((condition.equalsIgnoreCase("null")) && (i < tmp.size() - 1)
          && (!allowedNullOnce)) {
        tmp.remove(i);
        tmp.add(node);
        i--;
        allowedNullOnce = true;
      } else {
        if (i == 0) {
          result = result
              + "\nif ("
              + (condition.equalsIgnoreCase("null") ? "/*insert condition*/"
                  : "/*" + condition + "*/") + ") ";
        } else if ((i == tmp.size() - 1)
            && (condition.equalsIgnoreCase("null"))) {
          result = result + "\nelse ";
        } else {
          result = result
              + "\nelse if("
              + (condition.equalsIgnoreCase("null") ? "/*insert condition*/"
                  : "/*" + condition + "*/") + ") ";
        }
        result = result + "addSubBehaviour(new " + node.getName()
            + "Behaviour(this.myAgent" + params + "));";
      }
    }
    return result;
  }

  public static boolean existTransitionToSelf(Node e, Model m) {
    boolean result = false;
    for (Iterator<Transition> iterator = m.getTransitions().iterator(); iterator
        .hasNext();) {
      Transition tmp = iterator.next();
      if ((tmp.getSource().getLabel().equalsIgnoreCase(e.getLabel()))
          && (tmp.getTarget().getLabel().equalsIgnoreCase(e
              .getLabel()))) {
        result = true;
      }
    }
    return result;
```

```
}

public static String getTransitionToSelfConditionOfChild(Node e, Model m) {
  String result = "true";
  for (Iterator<Transition> iterator = m.getTransitions().iterator(); iterator
      .hasNext();) {
    Transition tmp = iterator.next();
    if ((tmp.getSource().getLabel().equalsIgnoreCase(subBehavioursOf(e,
        m).get(0).getLabel()))
        && (tmp.getTarget().getLabel()
            .equalsIgnoreCase(subBehavioursOf(e, m).get(0)
                .getLabel()))) {
      result = (tmp.getTE().equalsIgnoreCase("null") ? "/*insert condition*/"
          : tmp.getTE());
    }
  }
  return result;
}

public static String getTransitionToChildOf(Node e, Model m) {
  String result = "true";
  for (Iterator<Transition> iterator = m.getTransitions().iterator(); iterator
      .hasNext();) {
    Transition tmp = iterator.next();
    if ((!(tmp.getSource().getLabel().equalsIgnoreCase(subBehavioursOf(
        e, m).get(0).getLabel())))
        && (tmp.getTarget().getLabel()
            .equalsIgnoreCase(subBehavioursOf(e, m).get(0)
                .getLabel()))) {
      result = tmp.getTE();
    }
  }
  return result;
}

public static String determineBehaviourType(Node e, Model m) {
  if (e.getType().equalsIgnoreCase("AND")) {
    System.out
        .print("\nFound a behaviour with name: " + e.getName()
            + " found of type: "
            + ComplexBehaviourHelper.TYPE_PARALLEL);
    return ComplexBehaviourHelper.TYPE_PARALLEL;
  } else if ((subNodesOf(e, m).size() == 2)
      && (existTransitionToSelf((sortSubNodes(subNodesOf(e, m)))
          .get(1), m))) {
    System.out.print("\nFound a behaviour with name: " + e.getName()
        + " found of type: " + ComplexBehaviourHelper.TYPE_FOREVER);
    return ComplexBehaviourHelper.TYPE_FOREVER;
  } else if ((subNodesOf(e, m).size() == 3)
      && (existTransitionToSelf((sortSubNodes(subNodesOf(e, m)))
          .get(1), m))) {
    System.out.print("\nFound a behaviour with name: " + e.getName()
        + " found of type: "
        + ComplexBehaviourHelper.TYPE_ONE_OR_MORE_TIMES);
    return ComplexBehaviourHelper.TYPE_ONE_OR_MORE_TIMES;
  } else if (existConditionInSubNodesOf(e, m)) {
    if (subNodesOf(e, m).size() == 4) {
      System.out.print("\nFound a behaviour with name: "
          + e.getName() + " found of type: "
          + ComplexBehaviourHelper.TYPE_ZERO_OR_MORE_TIMES);
      return ComplexBehaviourHelper.TYPE_ZERO_OR_MORE_TIMES;
    } else {
      System.out.print("\nFound a behaviour with name: "
          + e.getName() + " found of type: "
          + ComplexBehaviourHelper.TYPE_OR);
      return ComplexBehaviourHelper.TYPE_OR;
    }
  } else {
    System.out
        .print("\nFound a behaviour with name: " + e.getName()
            + " found of type: "
            + ComplexBehaviourHelper.TYPE_SEQUENCE);
    return ComplexBehaviourHelper.TYPE_SEQUENCE;
  }
}
```

```
public static String addVariables(Node e, Model m) {
  String result = new String();
  for (Iterator iterator = e.getVariables().iterator(); iterator
      .hasNext();) {
    Variable variable = (Variable) iterator.next();
    if (existVariableInParentNode(variable, e, m)) {
      result = result + "\t\t" + variable.getType() + "Holder "
          + variable.getName() + " = null;";
    } else {
      result = result + "\t\t" + variable.getType() + "Holder "
          + variable.getName() + " = new " + variable.getType()
          + "Holder(this);";
    }
  }
  return result;
}

public static String getParams(Node e, Model m) {
  String result = new String();
  for (Iterator iterator = e.getVariables().iterator(); iterator
      .hasNext();) {
    Variable variable = (Variable) iterator.next();
    if (existVariableInParentNode(variable, e, m)) {
      result = result + ", " + variable.getType() + "Holder "
          + variable.getName();
    }
  }
  return result;
}

public static String instantiateParams(Node e, Model m) {
  String result = new String();
  for (Iterator iterator = e.getVariables().iterator(); iterator
      .hasNext();) {
    Variable variable = (Variable) iterator.next();
    if (existVariableInParentNode(variable, e, m)) {
      result = result + "\t\t\tthis." + variable.getName() + " = "
          + variable.getName() + ";";
    }
  }
  return result;
}

public static String addMessageTemplateVariable(Node e) {
  if (e.getName().startsWith("Receive")) {
    return "protected MessageTemplate mt = null;";
  }
  return "";
}

public static String importMessageClasses(Node e) {
  String result = new String();
  if ((e.getName().startsWith("Receive"))
      || (e.getName().startsWith("Send"))) {
    result = result + "\n\timport jade.lang.acl.ACLMessage;";
  }
  if (e.getName().startsWith("Receive")) {
    result = result + "\n\timport jade.lang.acl.MessageTemplate;";
  }
  return result;
}

public static String addAction(Node e, Model m) {
  System.out.print("\nadding action for node with name: " + e.getName());
  String result = new String();
  if (e.getName().startsWith("Receive")) {
    for (Iterator<Transition> iterator = m.getTransitions().iterator(); iterator
        .hasNext();) {
      Transition tmp = iterator.next();
      if ((tmp.getSource().getLabel().equalsIgnoreCase(e.getLabel()))
          && (!(tmp.getTarget().getLabel().equalsIgnoreCase(e
              .getLabel())))) {
        Pattern messagePattern = Pattern
            .compile("[a-z]+\\([a-z,]+\\)");
        Matcher messageMatcher = messagePattern
            .matcher(getEventOfExpression(tmp.getTE()));
```

```
          boolean firstPerformative = true;
          while (messageMatcher.find()) {
            String nextEvent = messageMatcher.group();
            if (firstPerformative) {
              firstPerformative = false;
              result = result
                  + "\t\tmt = MessageTemplate.MatchPerformative(ACLMessage."
                  + nextEvent.substring(0,
                      nextEvent.indexOf("("))
                      .toUpperCase() + ");";
            } else {
              result = result
                  + "\n\t\tmt =
MessageTemplate.or(mt,MessageTemplate.MatchPerformative(ACLMessage."
                  + nextEvent.substring(0,
                      nextEvent.indexOf("("))
                      .toUpperCase() + "));";
            }
          }
          if (firstPerformative) {
            result = result
                + "\t\t/*insert MessageTemplate code here*/";
          }
          result = result
              + "\n\t\tACLMessage msg = myAgent.receive(mt);"
              + "\n\t\tif (msg != null) {"
              + "\n\t\t//insert message handling code"
              + "\n\t\t\tfinished = true;" + "\n\t\t}else {"
              + "\n\t\t\tblock();" + "\n\t\t}";
        }
      }
    } else if (e.getName().startsWith("Send")) {
      for (Iterator<Transition> iterator = m.getTransitions().iterator(); iterator
          .hasNext();) {
        Transition tmp = iterator.next();
        if ((tmp.getSource().getLabel().equalsIgnoreCase(e.getLabel()))
            && (!(tmp.getTarget().getLabel().equalsIgnoreCase(e
                .getLabel())))) {
          Pattern messagePattern = Pattern
              .compile("[a-z]+\\([a-z,]+\\)");
          Matcher messageMatcher = messagePattern
              .matcher(getEventOfExpression(tmp.getTE()));
          boolean firstPerformative = true;
          result = result + "ACLMessage msg = null;";
          while (messageMatcher.find()) {
            String nextEvent = messageMatcher.group();
            if (firstPerformative) {
              firstPerformative = false;
              result = result
                  + "\n\t\tif (/*insert condition*/) {"
                  + "\n\t\t\tmsg = new ACLMessage(ACLMessage."
                  + nextEvent.substring(0,
                      nextEvent.indexOf("("))
                      .toUpperCase() + ");" + "\n\t\t}";
            } else {
              result = result
                  + "\n\t\telse if (/*insert condition*/) {"
                  + "\n\t\t\tmsg = new ACLMessage(ACLMessage."
                  + nextEvent.substring(0,
                      nextEvent.indexOf("("))
                      .toUpperCase() + ");" + "\n\t\t}";
            }
          }
          result = result
              + "\n\t\t//insert message initialization code"
              + "\n\t\tmyAgent.send(msg);"
              + "\n\t\tfinished = true;";
        }
      }
    } else {
      if (e.getActivity() != null) {
        if (e.getActivity().startsWith("/*Java code*/")) {
          result = result + "\n\t\t" + e.getActivity().substring(13)
              + "\n\t\tfinished = true;";
        } else {
          result = result
```

```
            + "\n\t\t/*"
            + (e.getActivity().equalsIgnoreCase("null") ? "insert behaviour activity
code here"
                : e.getActivity()) + "*/"
            + "\n\t\tfinished = true;";
      }
    } else {
      result = result
          + "\n\t\t/*insert behaviour activity code here*/"
          + "\n\t\tfinished = true;";
    }
  }
  return result;
}

public static Node getProtocolParentNode(String performative, Node e,
    Model m) {
  // TO DO define the attributes
  for (Iterator iterator = m.getNodes().iterator(); iterator.hasNext();) {
    Node node = (Node) iterator.next();
    if ((e.getLabel().startsWith(node.getLabel()))
        && (e.getLabel().compareTo(node.getLabel()) != 0)) {
      for (Iterator iterator2 = node.getVariables().iterator(); iterator2
          .hasNext();) {
        Variable variable = (Variable) iterator2.next();
        if (variable.getName().equalsIgnoreCase(performative)) {
          return node;
        }
      }
    }
  }
  return null;
}

public static boolean existVariableInParentNode(Variable var, Node e,
    Model m) {
  boolean result = false;
  // TO DO define the attributes
  EList<Node> tmp = m.getNodes();
  for (Iterator iterator = tmp.iterator(); iterator.hasNext();) {
    Node node = (Node) iterator.next();
    if ((e.getLabel().startsWith(node.getLabel()))
        && (e.getLabel().compareTo(node.getLabel()) != 0)) {
      for (Iterator iterator2 = node.getVariables().iterator(); iterator2
          .hasNext();) {
        Variable variable = (Variable) iterator2.next();
        if (variable.getName().equalsIgnoreCase(var.getName())) {
          return true;
        }
      }
    }
  }
  return result;
}

public static EList<Node> subNodesOf(Node e, Model m) {
  EList<Node> results = new BasicEList<Node>();
  for (Iterator iterator = m.getNodes().iterator(); iterator.hasNext();) {
    Node node = (Node) iterator.next();
    if ((node.getLabel().startsWith(e.getLabel()))
        && (node.getLabel().length() == e.getLabel().length() + 2)) {
      results.add(node);
    }
  }
  return results;
}

public static EList<Node> subBehavioursOf(Node e, Model m) {
  EList<Node> results = new BasicEList<Node>();
  for (Iterator iterator = m.getNodes().iterator(); iterator.hasNext();) {
    Node node = (Node) iterator.next();
    if ((node.getLabel().startsWith(e.getLabel()))
        && (node.getLabel().length() == e.getLabel().length() + 2)
        && (node.getType().equalsIgnoreCase("AND")
            || (node.getType().equalsIgnoreCase("OR")) || (node
            .getType().equalsIgnoreCase("BASIC")))) {
```

```
        results.add(node);
      }
    }
    return results;
  }

  public static EList<Node> sortSubNodes(EList<Node> list) {
    int n = list.size();
    boolean doMore = true;
    while (doMore) {
      n--;
      doMore = false; // assume this is our last pass over the array
      for (int i = 0; i < n; i++) {
        if (Integer.parseInt(list.get(i).getLabel().substring(
            list.get(i).getLabel().lastIndexOf(".") + 1,
            list.get(i).getLabel().length())) > Integer
            .parseInt(list.get(i + 1).getLabel()
                .substring(
                    list.get(i + 1).getLabel().lastIndexOf(
                        ".") + 1,
                    list.get(i + 1).getLabel().length()))) {
          // exchange elements
          list.move(i, i + 1);
          doMore = true; // after an exchange, must look again
        }
      }
    }
    return list;
  }

  public static String getConditionOfExpression(String expression) {
    // pattern for conditions
    Pattern conditionPattern = Pattern

  .compile("^([\\w\\W&&[^/\\[\\]]]+)?(\\[[\\w\\W&&[^\\[\\]]]+\\])(/[\\w\\W]+)?$");
    Matcher conditionMatcher = conditionPattern.matcher(expression);
    if (conditionMatcher.find()
        && (conditionMatcher.group().length() == expression.length())) {
      StringTokenizer st = new StringTokenizer(expression, "]");
      String condition = st.nextToken();
      condition = condition.substring(condition.indexOf("[") + 1);
      return condition;
    }
    return null;
  }

  public static String getEventOfExpression(String expression) {
    // pattern for events
    Pattern eventPattern = Pattern
        .compile("^[\\w\\W&&[^/\\[\\]]]+(\\[[\\w\\W&&[^\\[\\]]]+\\])?(/[\\w\\W]+)?$");
    Matcher eventMatcher = eventPattern.matcher(expression);
    if (eventMatcher.find()
        && (eventMatcher.group().length() == expression.length())) {
      StringTokenizer st = new StringTokenizer(expression, "[]/");
      return st.nextToken();
    }
    return null;
  }

  public static String getActionOfExpression(String expression) {
    // pattern for actions
    Pattern actionPattern = Pattern

  .compile("^([\\w\\W&&[^/\\[\\]]]+)?(\\[[\\w\\W&&[^\\[\\]]]+\\])?(/[\\w\\W]+)$");
    Matcher actionMatcher = actionPattern.matcher(expression);
    if (actionMatcher.find()
        && (actionMatcher.group().length() == expression.length())) {
      String action = expression
          .substring(expression.lastIndexOf("/") + 1);
      return action;
    }
    return null;
  }
}
```

Annex 6.

The Meetings Management System Models

Listing 34. The SAG model in XML format (SAGModel.xmi file)

```
<?xml version="1.0" encoding="ASCII" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SAG="http://mi.parisdescartes.fr/ASEME/metamodels/SAG"
xsi:schemaLocation="http://mi.parisdescartes.fr/ASEME/metamodels/SAG
../metamodels/SAG.ecore">
  <SAG:Actor name="PersonalAssistant" my_goal="/1 /2 /3 /4" />
  <SAG:Goal name="RequestNewMeeting" depender="/0" dependee="/5" requirements="A new
meeting needs to be arranged" />
  <SAG:Goal name="RequestChangeMeeting" depender="/0" dependee="/5" requirements="The
meeting date needs to change" />
  <SAG:Goal name="NegotiateMeetingDate" depender="/0" dependee="/5" requirements="The
meeting date must match the preferences of the majority" />
  <SAG:Goal name="LearnUserHabits" depender="/0" requirements="Minimize interaction
with the user by learning his preferences on the meeting dates" />
  <SAG:Actor name="MeetingsManager" />
  <SAG:Actor name="User" my_goal="/7" />
  <SAG:Goal name="ManageMeetings" depender="/6" dependee="/0" />
</xmi:XMI>
```

Listing 35. The initial SUC model in XML format (SUCModelInitial.xmi file)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:SUC="http://mi.parisdescartes.fr/ASEME/metamodels/SUC">
  <SUC:UseCase name="RequestNewMeeting" interacter="/5 /6" specified_by="A new meeting
needs to be arranged" />
  <SUC:UseCase name="RequestChangeMeeting" interacter="/5 /6" specified_by="The
meeting date needs to change" />
  <SUC:UseCase name="NegotiateMeetingDate" interacter="/5 /6" specified_by="The
meeting date must match the preferences of the majority" />
  <SUC:UseCase name="LearnUserHabits" interacter="/5" specified_by="minimize
interaction with the user by learning his preferences on the meeting dates" />
  <SUC:UseCase name="ManageMeetings" interacter="/7 /5" />
```

```
  <SUC:Role name="PersonalAssistant" interacts_with="/0 /1 /2 /3 /4" />
  <SUC:Role name="MeetingsManager" interacts_with="/0 /1 /2" />
  <SUC:Role name="User" interacts_with="/4" />
</xmi:XMI>
```

Listing 36. The refined SUC model in XML format (SUCModelRefined.xmi file)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:SUC="http://mi.parisdescartes.fr/ASEME/metamodels/SUC"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mi.parisdescartes.fr/ASEME/metamodels/SUC
../metamodels/SUC.ecore">
  <SUC:UseCase name="RequestNewMeeting" interacter="/5 /6" specified_by="A new meeting
needs to be arranged" include="/8 /9" />
  <SUC:UseCase name="RequestChangeMeeting" interacter="/5 /6" specified_by="The
meeting date needs to change" include="/10 /11" />
  <SUC:UseCase name="NegotiateMeetingDate" interacter="/5 /6" specified_by="The
meeting date must match the preferences of the majority" include="/17 /18 /19 /20 /21"
/>
  <SUC:UseCase name="LearnUserHabits" interacter="/5" specified_by="minimize
interaction with the user by learning his preferences on the meeting dates"
include="/12 /13" />
  <SUC:UseCase name="ManageMeetings" interacter="/7 /5" include="/14 /15 /16" />
  <SUC:Role name="PersonalAssistant" interacts_with="/0 /1 /2 /3 /4 /8 /10 /20 /9 /11
/17 /18 /12 /13 /14 /15 /16 /19 /21" />
  <SUC:Role name="MeetingsManager" interacts_with="/0 /1 /2" />
  <SUC:Role name="User" interacts_with="/4" />
  <SUC:UseCase name="SendNewRequest" interacter="/5" specified_by="use the Agent
Platform MPI to send the ACL message" included_by="/0" />
  <SUC:UseCase name="ReceiveNewResults" interacter="/5" specified_by="use the Agent
Platform MPI to receive an ACL message" included_by="/0" />
  <SUC:UseCase name="SendChangeRequest" interacter="/5" specified_by="use the Agent
Platform MPI to send the ACL message" included_by="/1" />
  <SUC:UseCase name="ReceiveChangeResults" interacter="/5" specified_by="use the Agent
Platform MPI to receive an ACL message" included_by="/1" />
  <SUC:UseCase name="LearnUserPreference" interacter="/5" specified_by="use a simple
learning algorithm for the user's preference" included_by="/3" />
  <SUC:UseCase name="UpdateUserPreferences" interacter="/5" specified_by="update the
user preference file on disk" included_by="/3" />
  <SUC:UseCase name="GetUserRequest" interacter="/5" specified_by="the HMI sends a
request" included_by="/4" />
  <SUC:UseCase name="ReadSchedule" interacter="/5" specified_by="read the user's
schedule from the disk" included_by="/4" />
  <SUC:UseCase name="ShowResults" interacter="/5" specified_by="send a response to the
HMI regarding the user's request" included_by="/4" />
  <SUC:UseCase name="ReceiveProposedDate" interacter="/5" specified_by="use the Agent
Platform MPI to receive an ACL message" included_by="/2" />
  <SUC:UseCase name="ReceiveOutcome" interacter="/5" specified_by="use the Agent
Platform MPI to receive an ACL message" included_by="/2" />
  <SUC:UseCase name="UpdateSchedule" interacter="/5" specified_by="update the user
schedule file on disk" included_by="/2" />
  <SUC:UseCase name="SendResults" interacter="/5" specified_by="use the Agent Platform
MPI to send the ACL message" included_by="/2" />
  <SUC:UseCase name="DecideResponse" interacter="/5" specified_by="use a reasoning
technique to decide if the proposed date matches the user's profile" included_by="/2"
/>
</xmi:XMI>
```

Listing 37. The initial SRM model in XML format (SRMModelInitial.xmi file)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:SRM="http://mi.parisdescartes.fr/ASEME/metamodels/SRM"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://mi.parisdescartes.fr/ASEME/metamodels/SRM
../metamodels/SRM.ecore">
  <SRM:Role activities="/20 /3 /5 /15 /4 /6 /12 /13 /7 /8 /9 /10 /11 /14 /16"
name="PersonalAssistant" capabilities="/17 /18 /19 /21" />
  <SRM:Role name="MeetingsManager" capabilities="/17 /18 /19" />
  <SRM:Role name="User" capabilities="/21" />
```

```
  <SRM:Activity name="SendNewRequest" functionality="use the Agent Platform MPI to
send the ACL message" />
  <SRM:Activity name="ReceiveNewResults" functionality="use the Agent Platform MPI to
receive an ACL message" />
  <SRM:Activity name="SendChangeRequest" functionality="use the Agent Platform MPI to
send the ACL message" />
  <SRM:Activity name="ReceiveChangeResults" functionality="use the Agent Platform MPI
to receive an ACL message" />
  <SRM:Activity name="LearnUserPreference" functionality="use a simple learning
algorithm for the user's preference" />
  <SRM:Activity name="UpdateUserPreferences" functionality="update the user preference
file on disk" />
  <SRM:Activity name="GetUserRequest" functionality="the HMI sends a request" />
  <SRM:Activity name="ReadSchedule" functionality="read the user's schedule from the
disk" />
  <SRM:Activity name="ShowResults" functionality="send a response to the HMI regarding
the user's request" />
  <SRM:Activity name="ReceiveProposedDate" functionality="use the Agent Platform MPI
to receive an ACL message" />
  <SRM:Activity name="ReceiveOutcome" functionality="use the Agent Platform MPI to
receive an ACL message" />
  <SRM:Activity name="UpdateSchedule" functionality="update the user schedule file on
disk" />
  <SRM:Activity name="SendResults" functionality="use the Agent Platform MPI to send
the ACL message" />
  <SRM:Activity name="DecideResponse" functionality="use a reasoning technique to
decide if the proposed date matches the user's profile" />
  <SRM:Capability name="RequestNewMeeting" activities="/3 /4" />
  <SRM:Capability name="RequestChangeMeeting" activities="/5 /6" />
  <SRM:Capability name="NegotiateMeetingDate" activities="/12 /13 /14 /15 /16" />
  <SRM:Capability name="LearnUserHabits" activities="/7 /8" />
  <SRM:Capability name="ManageMeetings" activities="/9 /10 /11" />
</xmi:XMI>
```

Listing 38. The refined SRM model in XML format (SRMModelRefined.xmi file)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SRM="http://mi.parisdescartes.fr/ASEME/metamodels/SRM"
xsi:schemaLocation="http://mi.parisdescartes.fr/ASEME/metamodels/SRM
../metamodels/SRM.ecore">
  <SRM:Role xmi:id="_w1b_lVc3Ed6LDYeRFx0dIA" activities="_w1cmpFc3Ed6LDYeRFx0dIA
_w1b_mFc3Ed6LDYeRFx0dIA _w1b_m1c3Ed6LDYeRFx0dIA _w1b_pFc3Ed6LDYeRFx0dIA
_w1b_mVc3Ed6LDYeRFx0dIA _w1b_m1c3Ed6LDYeRFx0dIA _w1b_oVc3Ed6LDYeRFx0dIA
_w1b_o1c3Ed6LDYeRFx0dIA _w1b_nFc3Ed6LDYeRFx0dIA _w1b_nVc3Ed6LDYeRFx0dIA
_w1b_n1c3Ed6LDYeRFx0dIA _w1b_n1c3Ed6LDYeRFx0dIA _w1b_oFc3Ed6LDYeRFx0dIA
_w1b_o1c3Ed6LDYeRFx0dIA _w1cmoFc3Ed6LDYeRFx0dIA" protocols="_w1cmqFc3Ed6LDYeRFx0dIA
_w1cmp1c3Ed6LDYeRFx0dIA _w1cmp1c3Ed6LDYeRFx0dIA" liveness="personal assistant =
(manage meetings. learn user habits)ω || (negotiate meeting date)ω ↲ manage meetings =
get user request. (read schedule | request change meeting | request new meeting). show
results ↲ learn user habits = learn user preference. update user preferences ↲ request
change meeting = send change request. receive change results ↲ request new meeting =
send new request. receive new results ↲ negotiate meeting date = receive proposed
date. (decide response. send results. receive outcome)+. update schedule ↲"
name="PersonalAssistant" capabilities="_w1cmoVc3Ed6LDYeRFx0dIA _w1cmo1c3Ed6LDYeRFx0dIA
_w1cmo1c3Ed6LDYeRFx0dIA _w1cmpVc3Ed6LDYeRFx0dIA" />
  <SRM:Role xmi:id="_w1b_l1c3Ed6LDYeRFx0dIA" name="MeetingsManager"
capabilities="_w1cmoVc3Ed6LDYeRFx0dIA _w1cmo1c3Ed6LDYeRFx0dIA _w1cmo1c3Ed6LDYeRFx0dIA"
/>
  <SRM:Role xmi:id="_w1b_l1c3Ed6LDYeRFx0dIA" name="User"
capabilities="_w1cmpVc3Ed6LDYeRFx0dIA" />
  <SRM:Activity xmi:id="_w1b_mFc3Ed6LDYeRFx0dIA" name="SendNewRequest"
functionality="use the Agent Platform MPI to send the ACL message" />
  <SRM:Activity xmi:id="_w1b_mVc3Ed6LDYeRFx0dIA" name="ReceiveNewResults"
functionality="use the Agent Platform MPI to receive an ACL message" />
  <SRM:Activity xmi:id="_w1b_m1c3Ed6LDYeRFx0dIA" name="SendChangeRequest"
functionality="use the Agent Platform MPI to send the ACL message" />
  <SRM:Activity xmi:id="_w1b_m1c3Ed6LDYeRFx0dIA" name="ReceiveChangeResults"
functionality="use the Agent Platform MPI to receive an ACL message" />
  <SRM:Activity xmi:id="_w1b_nFc3Ed6LDYeRFx0dIA" name="LearnUserPreference"
functionality="use a simple learning algorithm for the user's preference" />
```

```
  <SRM:Activity xmi:id="_w1b_nVc3Ed6LDYeRFx0dIA" name="UpdateUserPreferences"
functionality="update the user preference file on disk" />
  <SRM:Activity xmi:id="_w1b_nlc3Ed6LDYeRFx0dIA" name="GetUserRequest"
functionality="the HMI sends a request" />
  <SRM:Activity xmi:id="_w1b_n1c3Ed6LDYeRFx0dIA" name="ReadSchedule"
functionality="read the user's schedule from the disk" />
  <SRM:Activity xmi:id="_w1b_oFc3Ed6LDYeRFx0dIA" name="ShowResults"
functionality="send a response to the HMI regarding the user's request" />
  <SRM:Activity xmi:id="_w1b_oVc3Ed6LDYeRFx0dIA" name="ReceiveProposedDate"
functionality="use the Agent Platform MPI to receive an ACL message" />
  <SRM:Activity xmi:id="_w1b_olc3Ed6LDYeRFx0dIA" name="ReceiveOutcome"
functionality="use the Agent Platform MPI to receive an ACL message" />
  <SRM:Activity xmi:id="_w1b_o1c3Ed6LDYeRFx0dIA" name="UpdateSchedule"
functionality="update the user schedule file on disk" />
  <SRM:Activity xmi:id="_w1b_pFc3Ed6LDYeRFx0dIA" name="SendResults" functionality="use
the Agent Platform MPI to send the ACL message" />
  <SRM:Activity xmi:id="_w1cmoFc3Ed6LDYeRFx0dIA" name="DecideResponse"
functionality="use a reasoning technique to decide if the proposed date matches the
user's profile" />
  <SRM:Capability xmi:id="_w1cmoVc3Ed6LDYeRFx0dIA" name="RequestNewMeeting"
activities="_w1b_mFc3Ed6LDYeRFx0dIA _w1b_mVc3Ed6LDYeRFx0dIA" />
  <SRM:Capability xmi:id="_w1cmolc3Ed6LDYeRFx0dIA" name="RequestChangeMeeting"
activities="_w1b_mlc3Ed6LDYeRFx0dIA _w1b_m1c3Ed6LDYeRFx0dIA" />
  <SRM:Capability xmi:id="_w1cmo1c3Ed6LDYeRFx0dIA" name="NegotiateMeetingDate"
activities="_w1b_oVc3Ed6LDYeRFx0dIA _w1b_olc3Ed6LDYeRFx0dIA _w1b_o1c3Ed6LDYeRFx0dIA
_w1b_pFc3Ed6LDYeRFx0dIA _w1cmoFc3Ed6LDYeRFx0dIA" />
  <SRM:Capability xmi:id="_w1cmpFc3Ed6LDYeRFx0dIA" name="LearnUserHabits"
activities="_w1b_nFc3Ed6LDYeRFx0dIA _w1b_nVc3Ed6LDYeRFx0dIA" />
  <SRM:Capability xmi:id="_w1cmpVc3Ed6LDYeRFx0dIA" name="ManageMeetings"
activities="_w1b_nlc3Ed6LDYeRFx0dIA _w1b_n1c3Ed6LDYeRFx0dIA _w1b_oFc3Ed6LDYeRFx0dIA"
/>
  <SRM:Protocol xmi:id="_w1cmplc3Ed6LDYeRFx0dIA" name="RequestNewMeeting">
    <participant>PersonalAssistant</participant>
  </SRM:Protocol>
  <SRM:Protocol xmi:id="_w1cmp1c3Ed6LDYeRFx0dIA" name="RequestChangeMeeting">
    <participant>PersonalAssistant</participant>
  </SRM:Protocol>
  <SRM:Protocol xmi:id="_w1cmqFc3Ed6LDYeRFx0dIA" name="NegotiateMeetingDate">
    <participant>PersonalAssistant</participant>
  </SRM:Protocol>
</xmi:XMI>
```

Listing 39. The intermediate Hutn text model (IAC.hutn file)

```
@Spec{
 Metamodel "IAC"{
  nsUri: "http://mi.parisdescartes.fr/ASEME/metamodels/IAC"
 }
}
IAC{
 Node "0.2.1.2.2.2.3"{
  type: "OR"
  name:
"_open_group_ReadSchedule_or_RequestChangeMeeting_or_RequestNewMeeting_close_group_"
  label: "0.2.1.2.2.2.3"
  activity: "null"
 }
 Node "0.2.1.2.2.2.2"{
  type: "BASIC"
  name: "GetUserRequest"
  label: "0.2.1.2.2.2.2"
  activity: "null"
 }
 Node "0.2.1.2.2.2.1"{
  type: "START"
  name: "0.2.1.2.2.2.1"
  label: "0.2.1.2.2.2.1"
  activity: "null"
 }
 Node "0.2.1.2.2"{
  type: "OR"
  name: "_open_group_ManageMeetings_sequence_LearnUserHabits_close_group_"
  label: "0.2.1.2.2"
```

```
  activity: "null"
 }
 Node "0.2.1.2.1"{
  type: "START"
  name: "0.2.1.2.1"
  label: "0.2.1.2.1"
  activity: "null"
 }
 Node "0.2.1.2.2.4"{
  type: "END"
  name: "0.2.1.2.2.4"
  label: "0.2.1.2.2.4"
  activity: "null"
 }
 Node "0.2.2.2.2.5"{
  type: "END"
  name: "0.2.2.2.2.5"
  label: "0.2.2.2.2.5"
  activity: "null"
 }
 Node "0.2.1.2.2.3"{
  type: "OR"
  name: "LearnUserHabits"
  label: "0.2.1.2.2.3"
  activity: "null"
 }
 Node "0.2.2.2.2.4"{
  type: "BASIC"
  name: "UpdateSchedule"
  label: "0.2.2.2.2.4"
  activity: "null"
 }
 Node "0.2.1.2.2.2"{
  type: "OR"
  name: "ManageMeetings"
  label: "0.2.1.2.2.2"
  activity: "null"
 }
 Node "0.2.2.2.2.3"{
  type: "OR"
  name:
"_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group__
one_or_more_times_"
  label: "0.2.2.2.2.3"
  activity: "null"
 }
 Node "0.2.1.2.2.1"{
  type: "START"
  name: "0.2.1.2.2.1"
  label: "0.2.1.2.2.1"
  activity: "null"
 }
 Node "0.2.2.2.2.2"{
  type: "BASIC"
  name: "ReceiveProposedDate"
  label: "0.2.2.2.2.2"
  activity: "null"
 }
 Node "0.2.2.2.2.1"{
  type: "START"
  name: "0.2.2.2.2.1"
  label: "0.2.2.2.2.1"
  activity: "null"
 }
 Node "0.2.1.2.2.2.3.6"{
  type: "END"
  name: "0.2.1.2.2.2.3.6"
  label: "0.2.1.2.2.2.3.6"
  activity: "null"
 }
 Node "0.2.1.2.2.2.3.5"{
  type: "OR"
  name: "RequestNewMeeting"
  label: "0.2.1.2.2.2.3.5"
  activity: "null"
 }
```

```
Node "0.2.1.2.2.2.3.4"{
 type: "OR"
 name: "RequestChangeMeeting"
 label: "0.2.1.2.2.2.3.4"
 activity: "null"
}
Node "0.2.1.2.2.2.3.3"{
 type: "BASIC"
 name: "ReadSchedule"
 label: "0.2.1.2.2.2.3.3"
 activity: "null"
}
Node "0.2.1.2.2.2.3.2"{
 type: "CONDITION"
 name: "0.2.1.2.2.2.3.2"
 label: "0.2.1.2.2.2.3.2"
 activity: "null"
}
Node "0.2.1.2.2.2.3.1"{
 type: "START"
 name: "0.2.1.2.2.2.3.1"
 label: "0.2.1.2.2.2.3.1"
 activity: "null"
}
Node "0.2.2.2.2.3.3"{
 type: "END"
 name: "0.2.2.2.2.3.3"
 label: "0.2.2.2.2.3.3"
 activity: "null"
}
Node "0.2.2.2.2.3.2"{
 type: "OR"
 name:
"_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group_"
 label: "0.2.2.2.2.3.2"
 activity: "null"
}
Node "0.2.2.2.2.3.1"{
 type: "START"
 name: "0.2.2.2.2.3.1"
 label: "0.2.2.2.2.3.1"
 activity: "null"
}
Node "0.2.2"{
 type: "OR"
 name: "0.2.2"
 label: "0.2.2"
 activity: "null"
}
Node "0.2.1"{
 type: "OR"
 name: "0.2.1"
 label: "0.2.1"
 activity: "null"
}
Node "0.2.2.2.2.3.2.5"{
 type: "END"
 name: "0.2.2.2.2.3.2.5"
 label: "0.2.2.2.2.3.2.5"
 activity: "null"
}
Node "0.2.2.2.2.3.2.4"{
 type: "BASIC"
 name: "ReceiveOutcome"
 label: "0.2.2.2.2.3.2.4"
 activity: "null"
}
Node "0.2.2.2.2.3.2.3"{
 type: "BASIC"
 name: "SendResults"
 label: "0.2.2.2.2.3.2.3"
 activity: "null"
}
Node "0.2.2.2.2.3.2.2"{
 type: "BASIC"
 name: "DecideResponse"
```

```
 label: "0.2.2.2.2.3.2.2"
 activity: "null"
}
Node "0"{
 type: "OR"
 name: "PersonalAssistant"
 label: "0"
 activity: "null"
}
Node "0.2.2.2.2.3.2.1"{
 type: "START"
 name: "0.2.2.2.2.3.2.1"
 label: "0.2.2.2.2.3.2.1"
 activity: "null"
}
Node "0.2.1.2.2.2.3.5.4"{
 type: "END"
 name: "0.2.1.2.2.2.3.5.4"
 label: "0.2.1.2.2.2.3.5.4"
 activity: "null"
}
Node "0.2.1.2.2.2.3.5.3"{
 type: "BASIC"
 name: "ReceiveNewResults"
 label: "0.2.1.2.2.2.3.5.3"
 activity: "null"
}
Node "0.2.2.3"{
 type: "END"
 name: "0.2.2.3"
 label: "0.2.2.3"
 activity: "null"
}
Node "0.2.1.2.2.2.3.5.2"{
 type: "BASIC"
 name: "SendNewRequest"
 label: "0.2.1.2.2.2.3.5.2"
 activity: "null"
}
Node "0.2.2.2"{
 type: "OR"
 name: "NegotiateMeetingDate_forever_"
 label: "0.2.2.2"
 activity: "null"
}
Node "0.2.1.2.2.2.3.5.1"{
 type: "START"
 name: "0.2.1.2.2.2.3.5.1"
 label: "0.2.1.2.2.2.3.5.1"
 activity: "null"
}
Node "0.2.2.1"{
 type: "START"
 name: "0.2.2.1"
 label: "0.2.2.1"
 activity: "null"
}
Node "0.2.2.2.2"{
 type: "OR"
 name: "NegotiateMeetingDate"
 label: "0.2.2.2.2"
 activity: "null"
}
Node "0.2.2.2.1"{
 type: "START"
 name: "0.2.2.2.1"
 label: "0.2.2.2.1"
 activity: "null"
}
Node "0.2.1.2.2.2.3.4.4"{
 type: "END"
 name: "0.2.1.2.2.2.3.4.4"
 label: "0.2.1.2.2.2.3.4.4"
 activity: "null"
}
Node "0.2.1.2.2.2.3.4.3"{
```

```
 type: "BASIC"
 name: "ReceiveChangeResults"
 label: "0.2.1.2.2.2.3.4.3"
 activity: "null"
}
Node "0.2.1.2.2.2.3.4.2"{
 type: "BASIC"
 name: "SendChangeRequest"
 label: "0.2.1.2.2.2.3.4.2"
 activity: "null"
}
Node "0.2.1.3"{
 type: "END"
 name: "0.2.1.3"
 label: "0.2.1.3"
 activity: "null"
}
Node "0.2.1.2"{
 type: "OR"
 name: "_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever_"
 label: "0.2.1.2"
 activity: "null"
}
Node "0.2.1.2.2.2.3.4.1"{
 type: "START"
 name: "0.2.1.2.2.2.3.4.1"
 label: "0.2.1.2.2.2.3.4.1"
 activity: "null"
}
Node "0.2.1.1"{
 type: "START"
 name: "0.2.1.1"
 label: "0.2.1.1"
 activity: "null"
}
Node "0.2.1.2.2.3.4"{
 type: "END"
 name: "0.2.1.2.2.3.4"
 label: "0.2.1.2.2.3.4"
 activity: "null"
}
Node "0.2.1.2.2.3.3"{
 type: "BASIC"
 name: "UpdateUserPreferences"
 label: "0.2.1.2.2.3.3"
 activity: "null"
}
Node "0.3"{
 type: "END"
 name: "0.3"
 label: "0.3"
 activity: "null"
}
Node "0.2.1.2.2.3.2"{
 type: "BASIC"
 name: "LearnUserPreference"
 label: "0.2.1.2.2.3.2"
 activity: "null"
}
Node "0.2"{
 type: "AND"
 name: "0.2"
 label: "0.2"
 activity: "null"
}
Node "0.2.1.2.2.3.1"{
 type: "START"
 name: "0.2.1.2.2.3.1"
 label: "0.2.1.2.2.3.1"
 activity: "null"
}
Node "0.1"{
 type: "START"
 name: "0.1"
 label: "0.1"
 activity: "null"
```

```
}
Node "0.2.1.2.2.2.5"{
 type: "END"
 name: "0.2.1.2.2.2.5"
 label: "0.2.1.2.2.2.5"
 activity: "null"
}
Node "0.2.1.2.2.2.4"{
 type: "BASIC"
 name: "ShowResults"
 label: "0.2.1.2.2.2.4"
 activity: "null"
}
Transition "0.2.2.2.2TO0.2.2.2.2"{
 name: "0.2.2.2.2TO0.2.2.2.2"
 TE: "null"
 source: Node "0.2.2.2.2"
 target: Node "0.2.2.2.2"
}
Transition "0.2.2.2TO0.2.2.3"{
 name: "0.2.2.2TO0.2.2.3"
 TE: "null"
 source: Node "0.2.2.2"
 target: Node "0.2.2.3"
}
Transition "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.5"{
 name: "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.5"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.2"
 target: Node "0.2.1.2.2.2.3.5"
}
Transition "0.2.2.2.2.1TO0.2.2.2.2.2"{
 name: "0.2.2.2.2.1TO0.2.2.2.2.2"
 TE: "null"
 source: Node "0.2.2.2.2.1"
 target: Node "0.2.2.2.2.2"
}
Transition "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.4"{
 name: "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.4"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.2"
 target: Node "0.2.1.2.2.2.3.4"
}
Transition "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.3"{
 name: "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.3"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.2"
 target: Node "0.2.1.2.2.2.3.3"
}
Transition "0.2.1.2.2.3.3TO0.2.1.2.2.3.4"{
 name: "0.2.1.2.2.3.3TO0.2.1.2.2.3.4"
 TE: "null"
 source: Node "0.2.1.2.2.3.3"
 target: Node "0.2.1.2.2.3.4"
}
Transition "0.2.1.2.2.1TO0.2.1.2.2.2"{
 name: "0.2.1.2.2.1TO0.2.1.2.2.2"
 TE: "null"
 source: Node "0.2.1.2.2.1"
 target: Node "0.2.1.2.2.2"
}
Transition "0.2.1.2.2.3.2TO0.2.1.2.2.3.3"{
 name: "0.2.1.2.2.3.2TO0.2.1.2.2.3.3"
 TE: "null"
 source: Node "0.2.1.2.2.3.2"
 target: Node "0.2.1.2.2.3.3"
}
Transition "0.2.1.2.2.3.1TO0.2.1.2.2.3.2"{
 name: "0.2.1.2.2.3.1TO0.2.1.2.2.3.2"
 TE: "null"
 source: Node "0.2.1.2.2.3.1"
 target: Node "0.2.1.2.2.3.2"
}
Transition "0.2.2.2.2.3.2.3TO0.2.2.2.2.3.2.4"{
 name: "0.2.2.2.2.3.2.3TO0.2.2.2.2.3.2.4"
 TE: "null"
```

```
 source: Node "0.2.2.2.2.3.2.3"
 target: Node "0.2.2.2.2.3.2.4"
}
Transition "0.2.2.2.2.4TO0.2.2.2.2.5"{
 name: "0.2.2.2.2.4TO0.2.2.2.2.5"
 TE: "null"
 source: Node "0.2.2.2.2.4"
 target: Node "0.2.2.2.2.5"
}
Transition "0.2.1.2.2TO0.2.1.2.2"{
 name: "0.2.1.2.2TO0.2.1.2.2"
 TE: "null"
 source: Node "0.2.1.2.2"
 target: Node "0.2.1.2.2"
}
Transition "0.2.1.2.2.2.3.5TO0.2.1.2.2.2.3.6"{
 name: "0.2.1.2.2.2.3.5TO0.2.1.2.2.2.3.6"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.5"
 target: Node "0.2.1.2.2.2.3.6"
}
Transition "0.2.1.1TO0.2.1.2"{
 name: "0.2.1.1TO0.2.1.2"
 TE: "null"
 source: Node "0.2.1.1"
 target: Node "0.2.1.2"
}
Transition "0.2.2.2.2.3.2.1TO0.2.2.2.2.3.2.2"{
 name: "0.2.2.2.2.3.2.1TO0.2.2.2.2.3.2.2"
 TE: "null"
 source: Node "0.2.2.2.2.3.2.1"
 target: Node "0.2.2.2.2.3.2.2"
}
Transition "0.1TO0.2"{
 name: "0.1TO0.2"
 TE: "null"
 source: Node "0.1"
 target: Node "0.2"
}
Transition "0.2.2.2.1TO0.2.2.2.2"{
 name: "0.2.2.2.1TO0.2.2.2.2"
 TE: "null"
 source: Node "0.2.2.2.1"
 target: Node "0.2.2.2.2"
}
Transition "0.2.1.2.2.2.3.3TO0.2.1.2.2.2.3.6"{
 name: "0.2.1.2.2.2.3.3TO0.2.1.2.2.2.3.6"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.3"
 target: Node "0.2.1.2.2.2.3.6"
}
Transition "0.2.1.2.2.2.3.5.3TO0.2.1.2.2.2.3.5.4"{
 name: "0.2.1.2.2.2.3.5.3TO0.2.1.2.2.2.3.5.4"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.5.3"
 target: Node "0.2.1.2.2.2.3.5.4"
}
Transition "0.2.2.2.2.3TO0.2.2.2.2.4"{
 name: "0.2.2.2.2.3TO0.2.2.2.2.4"
 TE: "null"
 source: Node "0.2.2.2.2.3"
 target: Node "0.2.2.2.2.4"
}
Transition "0.2.1.2.2.2.3.4.3TO0.2.1.2.2.2.3.4.4"{
 name: "0.2.1.2.2.2.3.4.3TO0.2.1.2.2.2.3.4.4"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.4.3"
 target: Node "0.2.1.2.2.2.3.4.4"
}
Transition "0.2.1.2TO0.2.1.3"{
 name: "0.2.1.2TO0.2.1.3"
 TE: "null"
 source: Node "0.2.1.2"
 target: Node "0.2.1.3"
}
Transition "0.2.1.2.2.2.3.5.2TO0.2.1.2.2.2.3.5.3"{
```

```
 name: "0.2.1.2.2.2.3.5.2TO0.2.1.2.2.2.3.5.3"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.5.2"
 target: Node "0.2.1.2.2.2.3.5.3"
}
Transition "0.2.1.2.2.2.3.1TO0.2.1.2.2.2.3.2"{
 name: "0.2.1.2.2.2.3.1TO0.2.1.2.2.2.3.2"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.1"
 target: Node "0.2.1.2.2.2.3.2"
}
Transition "0.2.1.2.2.3TO0.2.1.2.2.4"{
 name: "0.2.1.2.2.3TO0.2.1.2.2.4"
 TE: "null"
 source: Node "0.2.1.2.2.3"
 target: Node "0.2.1.2.2.4"
}
Transition "0.2.1.2.2.2.3.4.2TO0.2.1.2.2.2.3.4.3"{
 name: "0.2.1.2.2.2.3.4.2TO0.2.1.2.2.2.3.4.3"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.4.2"
 target: Node "0.2.1.2.2.2.3.4.3"
}
Transition "0.2.2.2.2.3.2.4TO0.2.2.2.2.3.2.5"{
 name: "0.2.2.2.2.3.2.4TO0.2.2.2.2.3.2.5"
 TE: "null"
 source: Node "0.2.2.2.2.3.2.4"
 target: Node "0.2.2.2.2.3.2.5"
}
Transition "0.2.1.2.2.2.3.5.1TO0.2.1.2.2.2.3.5.2"{
 name: "0.2.1.2.2.2.3.5.1TO0.2.1.2.2.2.3.5.2"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.5.1"
 target: Node "0.2.1.2.2.2.3.5.2"
}
Transition "0.2.1.2.1TO0.2.1.2.2"{
 name: "0.2.1.2.1TO0.2.1.2.2"
 TE: "null"
 source: Node "0.2.1.2.1"
 target: Node "0.2.1.2.2"
}
Transition "0.2.1.2.2.2.4TO0.2.1.2.2.2.5"{
 name: "0.2.1.2.2.2.4TO0.2.1.2.2.2.5"
 TE: "null"
 source: Node "0.2.1.2.2.2.4"
 target: Node "0.2.1.2.2.2.5"
}
Transition "0.2.1.2.2.2.3.4.1TO0.2.1.2.2.2.3.4.2"{
 name: "0.2.1.2.2.2.3.4.1TO0.2.1.2.2.2.3.4.2"
 TE: "null"
 source: Node "0.2.1.2.2.2.3.4.1"
 target: Node "0.2.1.2.2.2.3.4.2"
}
Transition "0.2.2.1TO0.2.2.2"{
 name: "0.2.2.1TO0.2.2.2"
 TE: "null"
 source: Node "0.2.2.1"
 target: Node "0.2.2.2"
}
Transition "0.2.2.2.2.2TO0.2.2.2.2.3"{
 name: "0.2.2.2.2.2TO0.2.2.2.2.3"
 TE: "null"
 source: Node "0.2.2.2.2.2"
 target: Node "0.2.2.2.2.3"
}
Transition "0.2.1.2.2.2.3TO0.2.1.2.2.2.4"{
 name: "0.2.1.2.2.2.3TO0.2.1.2.2.2.4"
 TE: "null"
 source: Node "0.2.1.2.2.2.3"
 target: Node "0.2.1.2.2.2.4"
}
Transition "0.2.2.2.2.3.2.2TO0.2.2.2.2.3.2.3"{
 name: "0.2.2.2.2.3.2.2TO0.2.2.2.2.3.2.3"
 TE: "null"
 source: Node "0.2.2.2.2.3.2.2"
 target: Node "0.2.2.2.2.3.2.3"
```

```
 }
 Transition "0.2.2.2.2.3.2TO0.2.2.2.2.3.3"{
  name: "0.2.2.2.2.3.2TO0.2.2.2.2.3.3"
  TE: "null"
  source: Node "0.2.2.2.2.3.2"
  target: Node "0.2.2.2.2.3.3"
 }
 Transition "0.2.1.2.2.2.2TO0.2.1.2.2.2.3"{
  name: "0.2.1.2.2.2.2TO0.2.1.2.2.2.3"
  TE: "null"
  source: Node "0.2.1.2.2.2.2"
  target: Node "0.2.1.2.2.2.3"
 }
 Transition "0.2.2.2.2.3.2TO0.2.2.2.2.3.2"{
  name: "0.2.2.2.2.3.2TO0.2.2.2.2.3.2"
  TE: "null"
  source: Node "0.2.2.2.2.3.2"
  target: Node "0.2.2.2.2.3.2"
 }
 Transition "0.2.2.2.2.3.1TO0.2.2.2.2.3.2"{
  name: "0.2.2.2.2.3.1TO0.2.2.2.2.3.2"
  TE: "null"
  source: Node "0.2.2.2.2.3.1"
  target: Node "0.2.2.2.2.3.2"
 }
 Transition "0.2.1.2.2.2.1TO0.2.1.2.2.2.2"{
  name: "0.2.1.2.2.2.1TO0.2.1.2.2.2.2"
  TE: "null"
  source: Node "0.2.1.2.2.2.1"
  target: Node "0.2.1.2.2.2.2"
 }
 Transition "0.2.1.2.2.2TO0.2.1.2.2.3"{
  name: "0.2.1.2.2.2TO0.2.1.2.2.3"
  TE: "null"
  source: Node "0.2.1.2.2.2"
  target: Node "0.2.1.2.2.3"
 }
 Transition "0.2.1.2.2.2.3.4TO0.2.1.2.2.2.3.6"{
  name: "0.2.1.2.2.2.3.4TO0.2.1.2.2.2.3.6"
  TE: "null"
  source: Node "0.2.1.2.2.2.3.4"
  target: Node "0.2.1.2.2.2.3.6"
 }
 Transition "0.2TO0.3"{
  name: "0.2TO0.3"
  TE: "null"
  source: Node "0.2"
  target: Node "0.3"
 }
 Model "fr.parisdescartes.mi.meetingsmanagement"{
  name: "fr.parisdescartes.mi.meetingsmanagement"
  nodes: Node "0.2.1.2.2.2.3", Node "0.2.1.2.2.2.2", Node "0.2.1.2.2.2.1", Node
"0.2.1.2.2", Node "0.2.1.2.1", Node "0.2.1.2.2.4", Node "0.2.2.2.2.5", Node
"0.2.1.2.2.3", Node "0.2.2.2.2.4", Node "0.2.1.2.2.2", Node "0.2.2.2.2.3", Node
"0.2.1.2.2.1", Node "0.2.2.2.2.2", Node "0.2.2.2.2.1", Node "0.2.1.2.2.2.3.6", Node
"0.2.1.2.2.2.3.5", Node "0.2.1.2.2.2.3.4", Node "0.2.1.2.2.2.3.3", Node
"0.2.1.2.2.2.3.2", Node "0.2.1.2.2.2.3.1", Node "0.2.2.2.2.3.3", Node "0.2.2.2.2.3.2",
Node "0.2.2.2.2.3.1", Node "0.2.2.2", Node "0.2.1", Node "0.2.2.2.2.3.2.5", Node
"0.2.2.2.2.3.2.4", Node "0.2.2.2.2.3.2.3", Node "0.2.2.2.2.3.2.2", Node "0", Node
"0.2.2.2.2.3.2.1", Node "0.2.1.2.2.2.3.5.4", Node "0.2.1.2.2.2.3.5.3", Node "0.2.2.3",
Node "0.2.1.2.2.2.3.5.2", Node "0.2.2.2", Node "0.2.1.2.2.2.3.5.1", Node "0.2.2.1",
Node "0.2.2.2.2", Node "0.2.2.2.1", Node "0.2.1.2.2.2.3.4.4", Node
"0.2.1.2.2.2.3.4.3", Node "0.2.1.2.2.2.3.4.2", Node "0.2.1.3", Node "0.2.1.2", Node
"0.2.1.2.2.2.3.4.1", Node "0.2.1.1", Node "0.2.1.2.2.3.4", Node "0.2.1.2.2.3.3", Node
"0.3", Node "0.2.1.2.2.3.2", Node "0.2", Node "0.2.1.2.2.3.1", Node "0.1", Node
"0.2.1.2.2.2.5", Node "0.2.1.2.2.2.4"
  transitions: Transition "0.2.2.2.2TO0.2.2.2.2", Transition "0.2.2.2TO0.2.2.3",
Transition "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.5", Transition "0.2.2.2.2.1TO0.2.2.2.2.2",
Transition "0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.4", Transition
"0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.3", Transition "0.2.1.2.2.3.3TO0.2.1.2.2.3.4",
Transition "0.2.1.2.2.1TO0.2.1.2.2.2", Transition "0.2.1.2.2.3.2TO0.2.1.2.2.3.3",
Transition "0.2.1.2.2.3.1TO0.2.1.2.2.3.2", Transition
"0.2.2.2.2.3.2.3TO0.2.2.2.2.3.2.4", Transition "0.2.2.2.2.4TO0.2.2.2.2.5", Transition
"0.2.1.2.2TO0.2.1.2.2", Transition "0.2.1.2.2.2.3.5TO0.2.1.2.2.2.3.6", Transition
"0.2.1.1TO0.2.1.2", Transition "0.2.2.2.2.3.2.1TO0.2.2.2.2.3.2.2", Transition
"0.1TO0.2", Transition "0.2.2.2.1TO0.2.2.2.2", Transition
```

```
"0.2.1.2.2.2.3.3TO0.2.1.2.2.2.3.6", Transition "0.2.1.2.2.2.3.5.3TO0.2.1.2.2.2.3.5.4",
Transition "0.2.2.2.2.3TO0.2.2.2.2.4", Transition
"0.2.1.2.2.2.3.4.3TO0.2.1.2.2.2.3.4.4", Transition "0.2.1.2TO0.2.1.3", Transition
"0.2.1.2.2.2.3.5.2TO0.2.1.2.2.2.3.5.3", Transition "0.2.1.2.2.2.3.1TO0.2.1.2.2.2.3.2",
Transition "0.2.1.2.2.3TO0.2.1.2.2.4", Transition
"0.2.1.2.2.2.3.4.2TO0.2.1.2.2.2.3.4.3", Transition "0.2.2.2.2.3.2.4TO0.2.2.2.2.3.2.5",
Transition "0.2.1.2.2.2.3.5.1TO0.2.1.2.2.2.3.5.2", Transition "0.2.1.2.1TO0.2.1.2.2",
Transition "0.2.1.2.2.2.4TO0.2.1.2.2.2.5", Transition
"0.2.1.2.2.2.3.4.1TO0.2.1.2.2.2.3.4.2", Transition "0.2.2.1TO0.2.2.2", Transition
"0.2.2.2.2.2TO0.2.2.2.2.3", Transition "0.2.1.2.2.2.3TO0.2.1.2.2.2.4", Transition
"0.2.2.2.2.3.2.2TO0.2.2.2.2.3.2.3", Transition "0.2.2.2.2.3.2TO0.2.2.2.2.3.3",
Transition "0.2.1.2.2.2.2TO0.2.1.2.2.2.3", Transition "0.2.2.2.2.3.2TO0.2.2.2.2.3.2",
Transition "0.2.2.2.2.3.1TO0.2.2.2.2.3.2", Transition "0.2.1.2.2.2.1TO0.2.1.2.2.2.2",
Transition "0.2.1.2.2.2TO0.2.1.2.2.3", Transition "0.2.1.2.2.2.3.4TO0.2.1.2.2.2.3.6",
Transition "0.2TO0.3"
 }
}
```

Listing 40. The initial IAC model in XML format (IACModelInitial.model file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:IAC="http://mi.parisdescartes.fr/ASEME/metamodels/IAC">
  <IAC:Node
name="_open_group_ReadSchedule_or_RequestChangeMeeting_or_RequestNewMeeting_close_grou
p_" type="OR" label="0.2.1.2.2.2.3" activity="null" />
  <IAC:Node name="GetUserRequest" type="BASIC" label="0.2.1.2.2.2.2" activity="null"
/>
  <IAC:Node name="0.2.1.2.2.2.1" type="START" label="0.2.1.2.2.2.1" activity="null" />
  <IAC:Node name="_open_group_ManageMeetings_sequence_LearnUserHabits_close_group_"
type="OR" label="0.2.1.2.2" activity="null" />
  <IAC:Node name="0.2.1.2.1" type="START" label="0.2.1.2.1" activity="null" />
  <IAC:Node name="0.2.1.2.2.4" type="END" label="0.2.1.2.2.4" activity="null" />
  <IAC:Node name="0.2.2.2.2.5" type="END" label="0.2.2.2.2.5" activity="null" />
  <IAC:Node name="LearnUserHabits" type="OR" label="0.2.1.2.2.3" activity="null" />
  <IAC:Node name="UpdateSchedule" type="BASIC" label="0.2.2.2.2.4" activity="null" />
  <IAC:Node name="ManageMeetings" type="OR" label="0.2.1.2.2.2" activity="null" />
  <IAC:Node
name="_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_gr
oup__one_or_more_times_" type="OR" label="0.2.2.2.2.3" activity="null" />
  <IAC:Node name="0.2.1.2.2.1" type="START" label="0.2.1.2.2.1" activity="null" />
  <IAC:Node name="ReceiveProposedDate" type="BASIC" label="0.2.2.2.2.2"
activity="null" />
  <IAC:Node name="0.2.2.2.2.1" type="START" label="0.2.2.2.2.1" activity="null" />
  <IAC:Node name="0.2.1.2.2.2.3.6" type="END" label="0.2.1.2.2.2.3.6" activity="null"
/>
  <IAC:Node name="RequestNewMeeting" type="OR" label="0.2.1.2.2.2.3.5" activity="null"
/>
  <IAC:Node name="RequestChangeMeeting" type="OR" label="0.2.1.2.2.2.3.4"
activity="null" />
  <IAC:Node name="ReadSchedule" type="BASIC" label="0.2.1.2.2.2.3.3" activity="null"
/>
  <IAC:Node name="0.2.1.2.2.2.3.2" type="CONDITION" label="0.2.1.2.2.2.3.2"
activity="null" />
  <IAC:Node name="0.2.1.2.2.2.3.1" type="START" label="0.2.1.2.2.2.3.1"
activity="null" />
  <IAC:Node name="0.2.2.2.2.3.3" type="END" label="0.2.2.2.2.3.3" activity="null" />
  <IAC:Node
name="_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_gr
oup_" type="OR" label="0.2.2.2.2.3.2" activity="null" />
  <IAC:Node name="0.2.2.2.2.3.1" type="START" label="0.2.2.2.2.3.1" activity="null" />
  <IAC:Node name="0.2.2" type="OR" label="0.2.2" activity="null" />
  <IAC:Node name="0.2.1" type="OR" label="0.2.1" activity="null" />
  <IAC:Node name="0.2.2.2.2.3.2.5" type="END" label="0.2.2.2.2.3.2.5" activity="null"
/>
  <IAC:Node name="ReceiveOutcome" type="BASIC" label="0.2.2.2.2.3.2.4" activity="null"
/>
  <IAC:Node name="SendResults" type="BASIC" label="0.2.2.2.2.3.2.3" activity="null" />
  <IAC:Node name="DecideResponse" type="BASIC" label="0.2.2.2.2.3.2.2" activity="null"
/>
  <IAC:Node name="PersonalAssistant" type="OR" label="0" activity="null" />
  <IAC:Node name="0.2.2.2.2.3.2.1" type="START" label="0.2.2.2.2.3.2.1"
activity="null" />
```

```
  <IAC:Node name="0.2.1.2.2.2.3.5.4" type="END" label="0.2.1.2.2.2.3.5.4"
activity="null" />
  <IAC:Node name="ReceiveNewResults" type="BASIC" label="0.2.1.2.2.2.3.5.3"
activity="null" />
  <IAC:Node name="0.2.2.3" type="END" label="0.2.2.3" activity="null" />
  <IAC:Node name="SendNewRequest" type="BASIC" label="0.2.1.2.2.2.3.5.2"
activity="null" />
  <IAC:Node name="NegotiateMeetingDate_forever_" type="OR" label="0.2.2.2"
activity="null" />
  <IAC:Node name="0.2.1.2.2.2.3.5.1" type="START" label="0.2.1.2.2.2.3.5.1"
activity="null" />
  <IAC:Node name="0.2.2.1" type="START" label="0.2.2.1" activity="null" />
  <IAC:Node name="NegotiateMeetingDate" type="OR" label="0.2.2.2.2" activity="null" />
  <IAC:Node name="0.2.2.2.1" type="START" label="0.2.2.2.1" activity="null" />
  <IAC:Node name="0.2.1.2.2.2.3.4.4" type="END" label="0.2.1.2.2.2.3.4.4"
activity="null" />
  <IAC:Node name="ReceiveChangeResults" type="BASIC" label="0.2.1.2.2.2.3.4.3"
activity="null" />
  <IAC:Node name="SendChangeRequest" type="BASIC" label="0.2.1.2.2.2.3.4.2"
activity="null" />
  <IAC:Node name="0.2.1.3" type="END" label="0.2.1.3" activity="null" />
  <IAC:Node
name="_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever_"
type="OR" label="0.2.1.2" activity="null" />
  <IAC:Node name="0.2.1.2.2.2.3.4.1" type="START" label="0.2.1.2.2.2.3.4.1"
activity="null" />
  <IAC:Node name="0.2.1.1" type="START" label="0.2.1.1" activity="null" />
  <IAC:Node name="0.2.1.2.2.3.4" type="END" label="0.2.1.2.2.3.4" activity="null" />
  <IAC:Node name="UpdateUserPreferences" type="BASIC" label="0.2.1.2.2.3.3"
activity="null" />
  <IAC:Node name="0.3" type="END" label="0.3" activity="null" />
  <IAC:Node name="LearnUserPreference" type="BASIC" label="0.2.1.2.2.3.2"
activity="null" />
  <IAC:Node name="0.2" type="AND" label="0.2" activity="null" />
  <IAC:Node name="0.2.1.2.2.3.1" type="START" label="0.2.1.2.2.3.1" activity="null" />
  <IAC:Node name="0.1" type="START" label="0.1" activity="null" />
  <IAC:Node name="0.2.1.2.2.2.5" type="END" label="0.2.1.2.2.2.5" activity="null" />
  <IAC:Node name="ShowResults" type="BASIC" label="0.2.1.2.2.2.4" activity="null" />
  <IAC:Transition TE="null" source="#/38" target="#/38" name="0.2.2.2.2TO0.2.2.2.2" />
  <IAC:Transition TE="null" source="#/35" target="#/33" name="0.2.2.2TO0.2.2.3" />
  <IAC:Transition TE="null" source="#/18" target="#/15"
name="0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.5" />
  <IAC:Transition TE="null" source="#/13" target="#/12"
name="0.2.2.2.2.1TO0.2.2.2.2.2" />
  <IAC:Transition TE="null" source="#/18" target="#/16"
name="0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.4" />
  <IAC:Transition TE="null" source="#/18" target="#/17"
name="0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.3" />
  <IAC:Transition TE="null" source="#/48" target="#/47"
name="0.2.1.2.2.3.3TO0.2.1.2.2.3.4" />
  <IAC:Transition TE="null" source="#/11" target="#/9" name="0.2.1.2.2.1TO0.2.1.2.2.2"
/>
  <IAC:Transition TE="null" source="#/50" target="#/48"
name="0.2.1.2.2.3.2TO0.2.1.2.2.3.3" />
  <IAC:Transition TE="null" source="#/52" target="#/50"
name="0.2.1.2.2.3.1TO0.2.1.2.2.3.2" />
  <IAC:Transition TE="null" source="#/27" target="#/26"
name="0.2.2.2.2.3.2.3TO0.2.2.2.2.3.2.4" />
  <IAC:Transition TE="null" source="#/8" target="#/6" name="0.2.2.2.2.4TO0.2.2.2.2.5"
/>
  <IAC:Transition TE="null" source="#/3" target="#/3" name="0.2.1.2.2TO0.2.1.2.2" />
  <IAC:Transition TE="null" source="#/15" target="#/14"
name="0.2.1.2.2.2.3.5TO0.2.1.2.2.2.3.6" />
  <IAC:Transition TE="null" source="#/46" target="#/44" name="0.2.1.1TO0.2.1.2" />
  <IAC:Transition TE="null" source="#/30" target="#/28"
name="0.2.2.2.2.3.2.1TO0.2.2.2.2.3.2.2" />
  <IAC:Transition TE="null" source="#/53" target="#/51" name="0.1TO0.2" />
  <IAC:Transition TE="null" source="#/39" target="#/38" name="0.2.2.2.1TO0.2.2.2.2" />
  <IAC:Transition TE="null" source="#/17" target="#/14"
name="0.2.1.2.2.2.3.3TO0.2.1.2.2.2.3.6" />
  <IAC:Transition TE="null" source="#/32" target="#/31"
name="0.2.1.2.2.2.3.5.3TO0.2.1.2.2.2.3.5.4" />
  <IAC:Transition TE="null" source="#/10" target="#/8" name="0.2.2.2.2.3TO0.2.2.2.2.4"
/>
  <IAC:Transition TE="null" source="#/41" target="#/40"
name="0.2.1.2.2.2.3.4.3TO0.2.1.2.2.2.3.4.4" />
```

```
  <IAC:Transition TE="null" source="#/44" target="#/43" name="0.2.1.2TO0.2.1.3" />
  <IAC:Transition TE="null" source="#/34" target="#/32"
name="0.2.1.2.2.2.3.5.2TO0.2.1.2.2.2.3.5.3" />
  <IAC:Transition TE="null" source="#/19" target="#/18"
name="0.2.1.2.2.2.3.1TO0.2.1.2.2.2.3.2" />
  <IAC:Transition TE="null" source="#/7" target="#/5" name="0.2.1.2.2.3TO0.2.1.2.2.4"
/>
  <IAC:Transition TE="null" source="#/42" target="#/41"
name="0.2.1.2.2.2.3.4.2TO0.2.1.2.2.2.3.4.3" />
  <IAC:Transition TE="null" source="#/26" target="#/25"
name="0.2.2.2.2.3.2.4TO0.2.2.2.2.3.2.5" />
  <IAC:Transition TE="null" source="#/36" target="#/34"
name="0.2.1.2.2.2.3.5.1TO0.2.1.2.2.2.3.5.2" />
  <IAC:Transition TE="null" source="#/4" target="#/3" name="0.2.1.2.1TO0.2.1.2.2" />
  <IAC:Transition TE="null" source="#/55" target="#/54"
name="0.2.1.2.2.2.4TO0.2.1.2.2.2.5" />
  <IAC:Transition TE="null" source="#/45" target="#/42"
name="0.2.1.2.2.2.3.4.1TO0.2.1.2.2.2.3.4.2" />
  <IAC:Transition TE="null" source="#/37" target="#/35" name="0.2.2.1TO0.2.2.2" />
  <IAC:Transition TE="null" source="#/12" target="#/10"
name="0.2.2.2.2.2TO0.2.2.2.2.3" />
  <IAC:Transition TE="null" source="#/0" target="#/55"
name="0.2.1.2.2.2.3TO0.2.1.2.2.2.4" />
  <IAC:Transition TE="null" source="#/28" target="#/27"
name="0.2.2.2.2.3.2.2TO0.2.2.2.2.3.2.3" />
  <IAC:Transition TE="null" source="#/21" target="#/20"
name="0.2.2.2.2.3.2TO0.2.2.2.2.3.3" />
  <IAC:Transition TE="null" source="#/1" target="#/0"
name="0.2.1.2.2.2.2TO0.2.1.2.2.2.3" />
  <IAC:Transition TE="null" source="#/21" target="#/21"
name="0.2.2.2.2.3.2TO0.2.2.2.2.3.2" />
  <IAC:Transition TE="null" source="#/22" target="#/21"
name="0.2.2.2.2.3.1TO0.2.2.2.2.3.2" />
  <IAC:Transition TE="null" source="#/2" target="#/1"
name="0.2.1.2.2.2.1TO0.2.1.2.2.2.2" />
  <IAC:Transition TE="null" source="#/9" target="#/7" name="0.2.1.2.2.2TO0.2.1.2.2.3"
/>
  <IAC:Transition TE="null" source="#/16" target="#/14"
name="0.2.1.2.2.2.3.4TO0.2.1.2.2.2.3.6" />
  <IAC:Transition TE="null" source="#/51" target="#/49" name="0.2TO0.3" />
  <IAC:Model nodes="#/0 #/1 #/2 #/3 #/4 #/5 #/6 #/7 #/8 #/9 #/10 #/11 #/12 #/13 #/14
#/15 #/16 #/17 #/18 #/19 #/20 #/21 #/22 #/23 #/24 #/25 #/26 #/27 #/28 #/29 #/30 #/31
#/32 #/33 #/34 #/35 #/36 #/37 #/38 #/39 #/40 #/41 #/42 #/43 #/44 #/45 #/46 #/47 #/48
#/49 #/50 #/51 #/52 #/53 #/54 #/55" transitions="#/56 #/57 #/58 #/59 #/60 #/61 #/62
#/63 #/64 #/65 #/66 #/67 #/68 #/69 #/70 #/71 #/72 #/73 #/74 #/75 #/76 #/77 #/78 #/79
#/80 #/81 #/82 #/83 #/84 #/85 #/86 #/87 #/88 #/89 #/90 #/91 #/92 #/93 #/94 #/95 #/96
#/97 #/98 #/99" name="fr.parisdescartes.mi.meetingsmanagement" />
</xmi:XMI>
```

Listing 41. The refined IAC model in XML format (IACModelRefined.model file)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:IAC="http://mi.parisdescartes.fr/ASEME/metamodels/IAC">
  <IAC:Model xmi:id="_Phkl7VatEd6xWKSHAXqXJw" nodes="_PhiwsFatEd6xWKSHAXqXJw
_PhiwsVatEd6xWKSHAXqXJw _PhiwslatEd6xWKSHAXqXJw _Phiws1atEd6xWKSHAXqXJw
_PhiwtFatEd6xWKSHAXqXJw _PhiwtVatEd6xWKSHAXqXJw _PhiwtlatEd6xWKSHAXqXJw
_Phiwt1atEd6xWKSHAXqXJw _PhiwuFatEd6xWKSHAXqXJw _PhiwuVatEd6xWKSHAXqXJw
_PhiwulatEd6xWKSHAXqXJw _Phiwu1atEd6xWKSHAXqXJw _PhiwvFatEd6xWKSHAXqXJw
_PhiwvlatEd6xWKSHAXqXJw _Phiwv1atEd6xWKSHAXqXJw _PhiwwFatEd6xWKSHAXqXJw
_PhiwwVatEd6xWKSHAXqXJw _PhiwwlatEd6xWKSHAXqXJw _Phiww1atEd6xWKSHAXqXJw
_PhiwxFatEd6xWKSHAXqXJw _PhiwxVatEd6xWKSHAXqXJw _PhjXwFatEd6xWKSHAXqXJw
_PhjXwVatEd6xWKSHAXqXJw _PhjXwlatEd6xWKSHAXqXJw _PhjXw1atEd6xWKSHAXqXJw
_PhjXxFatEd6xWKSHAXqXJw _PhjXxVatEd6xWKSHAXqXJw _PhjXxlatEd6xWKSHAXqXJw
_PhjXx1atEd6xWKSHAXqXJw _PhjXyFatEd6xWKSHAXqXJw _PhjXyVatEd6xWKSHAXqXJw
_PhjXylatEd6xWKSHAXqXJw _PhjXy1atEd6xWKSHAXqXJw _PhjXzFatEd6xWKSHAXqXJw
_PhjXzVatEd6xWKSHAXqXJw _PhjXzlatEd6xWKSHAXqXJw _PhjXz1atEd6xWKSHAXqXJw
_PhjX0FatEd6xWKSHAXqXJw _PhjX0VatEd6xWKSHAXqXJw _PhjX0latEd6xWKSHAXqXJw
_PhjX01atEd6xWKSHAXqXJw _PhjX1FatEd6xWKSHAXqXJw _PhjX1VatEd6xWKSHAXqXJw
_PhjX11atEd6xWKSHAXqXJw _PhjX11atEd6xWKSHAXqXJw _PhjX2FatEd6xWKSHAXqXJw
_PhjX2VatEd6xWKSHAXqXJw _PhjX2latEd6xWKSHAXqXJw _PhjX21atEd6xWKSHAXqXJw
_PhjX3FatEd6xWKSHAXqXJw _PhjX3VatEd6xWKSHAXqXJw _PhjX31atEd6xWKSHAXqXJw
_PhjX31atEd6xWKSHAXqXJw _PhjX4FatEd6xWKSHAXqXJw _PhjX4VatEd6xWKSHAXqXJw
_PhjX4latEd6xWKSHAXqXJw" transitions="_Phj-0FatEd6xWKSHAXqXJw _Phj-0VatEd6xWKSHAXqXJw
```

```
_Phj-0latEd6xWKSHAXqXJw _Phj-01atEd6xWKSHAXqXJw _Phj-1FatEd6xWKSHAXqXJw _Phj-
1VatEd6xWKSHAXqXJw _Phj-1latEd6xWKSHAXqXJw _Phj-11atEd6xWKSHAXqXJw _Phj-
2FatEd6xWKSHAXqXJw _Phj-2VatEd6xWKSHAXqXJw _Phj-2latEd6xWKSHAXqXJw _Phj-
21atEd6xWKSHAXqXJw _Phj-3FatEd6xWKSHAXqXJw _Phj-3VatEd6xWKSHAXqXJw _Phj-
3latEd6xWKSHAXqXJw _Phj-31atEd6xWKSHAXqXJw _Phj-4FatEd6xWKSHAXqXJw _Phj-
4VatEd6xWKSHAXqXJw _Phj-4latEd6xWKSHAXqXJw _Phj-41atEd6xWKSHAXqXJw _Phj-
5FatEd6xWKSHAXqXJw _Phj-5VatEd6xWKSHAXqXJw _Phj-5latEd6xWKSHAXqXJw _Phj-
51atEd6xWKSHAXqXJw _Phj-6FatEd6xWKSHAXqXJw _Phj-6VatEd6xWKSHAXqXJw _Phj-
6latEd6xWKSHAXqXJw _Phj-61atEd6xWKSHAXqXJw _Phj-7FatEd6xWKSHAXqXJw _Phj-
7VatEd6xWKSHAXqXJw _Phj-7latEd6xWKSHAXqXJw _Phkl4FatEd6xWKSHAXqXJw
_Phkl4VatEd6xWKSHAXqXJw _Phkl41atEd6xWKSHAXqXJw _Phkl41atEd6xWKSHAXqXJw
_Phkl5FatEd6xWKSHAXqXJw _Phkl5VatEd6xWKSHAXqXJw _Phkl5latEd6xWKSHAXqXJw
_Phkl51atEd6xWKSHAXqXJw _Phkl6FatEd6xWKSHAXqXJw _Phkl6VatEd6xWKSHAXqXJw
_Phkl6latEd6xWKSHAXqXJw _Phkl61atEd6xWKSHAXqXJw _Phkl7FatEd6xWKSHAXqXJw"
name="fr.parisdescartes.mi.meetingsmanagement" variables="_PhiwvVatEd6xWKSHAXqXJw
proposeVar informVar acceptVar rejectVar" />
  <IAC:Node xmi:id="_PhiwsFatEd6xWKSHAXqXJw"
name="_open_group_ReadSchedule_or_RequestChangeMeeting_or_RequestNewMeeting_close_grou
p_" type="OR" label="0.2.1.2.2.3" activity="null" />
  <IAC:Node xmi:id="_PhiwsVatEd6xWKSHAXqXJw" name="GetUserRequest" type="BASIC"
label="0.2.1.2.2.2" activity="the HMI sends a request" />
  <IAC:Node xmi:id="_PhiwslatEd6xWKSHAXqXJw" name="0.2.1.2.2.1" type="START"
label="0.2.1.2.2.1" activity="null" />
  <IAC:Node xmi:id="_Phiws1atEd6xWKSHAXqXJw"
name="_open_group_ManageMeetings_sequence_LearnUserHabits_close_group_" type="OR"
label="0.2.1.2.2" activity="null" />
  <IAC:Node xmi:id="_PhiwtFatEd6xWKSHAXqXJw" name="0.2.1.2.1" type="START"
label="0.2.1.2.1" activity="null" />
  <IAC:Node xmi:id="_PhiwtVatEd6xWKSHAXqXJw" name="0.2.1.2.2.4" type="END"
label="0.2.1.2.2.4" activity="null" />
  <IAC:Node xmi:id="_PhiwtlatEd6xWKSHAXqXJw" name="0.2.2.2.2.5" type="END"
label="0.2.2.2.2.5" activity="null" />
  <IAC:Node xmi:id="_Phiwt1atEd6xWKSHAXqXJw" name="LearnUserHabits" type="OR"
label="0.2.1.2.2.3" activity="null" />
  <IAC:Node xmi:id="_PhiwuFatEd6xWKSHAXqXJw" name="UpdateSchedule" type="BASIC"
label="0.2.2.2.2.4" activity="update the user schedule file on disk"
variables="_PhiwvVatEd6xWKSHAXqXJw informVar" />
  <IAC:Node xmi:id="_PhiwuVatEd6xWKSHAXqXJw" name="ManageMeetings" type="OR"
label="0.2.1.2.2.2" activity="null" />
  <IAC:Node xmi:id="_PhiwulatEd6xWKSHAXqXJw"
name="_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_gr
oup__one_or_more_times_" type="OR" label="0.2.2.2.2.3" activity="null"
variables="_PhiwvVatEd6xWKSHAXqXJw acceptVar informVar proposeVar rejectVar" />
  <IAC:Node xmi:id="_Phiwu1atEd6xWKSHAXqXJw" name="0.2.1.2.2.1" type="START"
label="0.2.1.2.2.1" activity="null" />
  <IAC:Node xmi:id="_PhiwvFatEd6xWKSHAXqXJw" name="ReceiveProposedDate" type="BASIC"
label="0.2.2.2.2.2" activity="null" variables="_PhiwvVatEd6xWKSHAXqXJw proposeVar" />
  <IAC:Variable xmi:id="_PhiwvVatEd6xWKSHAXqXJw" name="e" type="Meeting" />
  <IAC:Variable xmi:id="proposeVar" name="propose" type="ACLMessage" />
  <IAC:Variable xmi:id="informVar" name="inform" type="ACLMessage" />
  <IAC:Variable xmi:id="acceptVar" name="accept" type="ACLMessage" />
  <IAC:Variable xmi:id="rejectVar" name="reject" type="ACLMessage" />
  <IAC:Node xmi:id="_PhiwvlatEd6xWKSHAXqXJw" name="0.2.2.2.2.1" type="START"
label="0.2.2.2.2.1" activity="null" />
  <IAC:Node xmi:id="_Phiwv1atEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.6" type="END"
label="0.2.1.2.2.2.3.6" activity="null" />
  <IAC:Node xmi:id="_PhiwwFatEd6xWKSHAXqXJw" name="RequestNewMeeting" type="OR"
label="0.2.1.2.2.2.3.5" activity="null" />
  <IAC:Node xmi:id="_PhiwwVatEd6xWKSHAXqXJw" name="RequestChangeMeeting" type="OR"
label="0.2.1.2.2.2.3.4" activity="null" />
  <IAC:Node xmi:id="_PhiwwlatEd6xWKSHAXqXJw" name="ReadSchedule" type="BASIC"
label="0.2.1.2.2.2.3.3" activity="read the user's schedule from the disk" />
  <IAC:Node xmi:id="_Phiww1atEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.2" type="CONDITION"
label="0.2.1.2.2.2.3.2" activity="null" />
  <IAC:Node xmi:id="_PhiwxFatEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.1" type="START"
label="0.2.1.2.2.2.3.1" activity="null" />
  <IAC:Node xmi:id="_PhiwxVatEd6xWKSHAXqXJw" name="0.2.2.2.2.3.3" type="END"
label="0.2.2.2.2.3.3" activity="null" />
  <IAC:Node xmi:id="_PhjXwFatEd6xWKSHAXqXJw"
name="_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_gr
oup_" type="OR" label="0.2.2.2.2.3.2" activity="null"
variables="_PhiwvVatEd6xWKSHAXqXJw acceptVar informVar proposeVar rejectVar" />
  <IAC:Node xmi:id="_PhjXwVatEd6xWKSHAXqXJw" name="0.2.2.2.2.3.1" type="START"
label="0.2.2.2.2.3.1" activity="null" />
```

```
  <IAC:Node xmi:id="_PhjXwlatEd6xWKSHAXqXJw"
name="NegotiateMeetingDate_forever__parallel_" type="OR" label="0.2.2" activity="null"
variables="_PhiwvVatEd6xWKSHAXqXJw" />
  <IAC:Node xmi:id="_PhjXw1atEd6xWKSHAXqXJw"
name="_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parall
el_" type="OR" label="0.2.1" activity="null" />
  <IAC:Node xmi:id="_PhjXxFatEd6xWKSHAXqXJw" name="0.2.2.2.2.3.2.5" type="END"
label="0.2.2.2.2.3.2.5" activity="null" />
  <IAC:Node xmi:id="_PhjXxVatEd6xWKSHAXqXJw" name="ReceiveOutcome" type="BASIC"
label="0.2.2.2.2.3.2.4" activity="null" variables="_PhiwvVatEd6xWKSHAXqXJw informVar
proposeVar" />
  <IAC:Node xmi:id="_PhjXxlatEd6xWKSHAXqXJw" name="SendResults" type="BASIC"
label="0.2.2.2.2.3.2.3" activity="null" variables="_PhiwvVatEd6xWKSHAXqXJw acceptVar
rejectVar" />
  <IAC:Node xmi:id="_PhjXx1atEd6xWKSHAXqXJw" name="DecideResponse" type="BASIC"
label="0.2.2.2.2.3.2.2" activity="use a reasoning technique to decide if the proposed
date matches the user's profile" variables="_PhiwvVatEd6xWKSHAXqXJw proposeVar" />
  <IAC:Node xmi:id="_PhjXyFatEd6xWKSHAXqXJw" name="PersonalAssistant" type="OR"
label="0" activity="null" />
  <IAC:Node xmi:id="_PhjXyVatEd6xWKSHAXqXJw" name="0.2.2.2.2.3.2.1" type="START"
label="0.2.2.2.2.3.2.1" activity="null" />
  <IAC:Node xmi:id="_PhjXylatEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.5.4" type="END"
label="0.2.1.2.2.2.3.5.4" activity="null" />
  <IAC:Node xmi:id="_PhjXy1atEd6xWKSHAXqXJw" name="ReceiveNewResults" type="BASIC"
label="0.2.1.2.2.2.3.5.3" activity="null" />
  <IAC:Node xmi:id="_PhjXzFatEd6xWKSHAXqXJw" name="0.2.2.3" type="END" label="0.2.2.3"
activity="null" />
  <IAC:Node xmi:id="_PhjXzVatEd6xWKSHAXqXJw" name="SendNewRequest" type="BASIC"
label="0.2.1.2.2.2.3.5.2" activity="null" />
  <IAC:Node xmi:id="_PhjXzlatEd6xWKSHAXqXJw" name="NegotiateMeetingDate_forever_"
type="OR" label="0.2.2.2" activity="null" variables="_PhiwvVatEd6xWKSHAXqXJw" />
  <IAC:Node xmi:id="_PhjXz1atEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.5.1" type="START"
label="0.2.1.2.2.2.3.5.1" activity="null" />
  <IAC:Node xmi:id="_PhjX0FatEd6xWKSHAXqXJw" name="0.2.2.1" type="START"
label="0.2.2.1" activity="null" />
  <IAC:Node xmi:id="_PhjX0VatEd6xWKSHAXqXJw" name="NegotiateMeetingDate" type="OR"
label="0.2.2.2.2" activity="null" variables="_PhiwvVatEd6xWKSHAXqXJw acceptVar
informVar proposeVar rejectVar" />
  <IAC:Node xmi:id="_PhjX0latEd6xWKSHAXqXJw" name="0.2.2.2.1" type="START"
label="0.2.2.2.1" activity="null" />
  <IAC:Node xmi:id="_PhjX01atEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.4.4" type="END"
label="0.2.1.2.2.2.3.4.4" activity="null" />
  <IAC:Node xmi:id="_PhjX1FatEd6xWKSHAXqXJw" name="ReceiveChangeResults" type="BASIC"
label="0.2.1.2.2.2.3.4.3" activity="null" />
  <IAC:Node xmi:id="_PhjX1VatEd6xWKSHAXqXJw" name="SendChangeRequest" type="BASIC"
label="0.2.1.2.2.2.3.4.2" activity="null" />
  <IAC:Node xmi:id="_PhjX1latEd6xWKSHAXqXJw" name="0.2.1.3" type="END" label="0.2.1.3"
activity="null" />
  <IAC:Node xmi:id="_PhjX11atEd6xWKSHAXqXJw"
name="_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever_"
type="OR" label="0.2.1.2" activity="null" />
  <IAC:Node xmi:id="_PhjX2FatEd6xWKSHAXqXJw" name="0.2.1.2.2.2.3.4.1" type="START"
label="0.2.1.2.2.2.3.4.1" activity="null" />
  <IAC:Node xmi:id="_PhjX2VatEd6xWKSHAXqXJw" name="0.2.1.1" type="START"
label="0.2.1.1" activity="null" />
  <IAC:Node xmi:id="_PhjX2latEd6xWKSHAXqXJw" name="0.2.1.2.2.3.4" type="END"
label="0.2.1.2.2.3.4" activity="null" />
  <IAC:Node xmi:id="_PhjX21atEd6xWKSHAXqXJw" name="UpdateUserPreferences" type="BASIC"
label="0.2.1.2.2.3.3" activity="update the user preference file on disk" />
  <IAC:Node xmi:id="_PhjX3FatEd6xWKSHAXqXJw" name="0.3" type="END" label="0.3"
activity="null" />
  <IAC:Node xmi:id="_PhjX3VatEd6xWKSHAXqXJw" name="LearnUserPreference" type="BASIC"
label="0.2.1.2.2.3.2" activity="use a simple learning algorithm for the user's
preference" />
  <IAC:Node xmi:id="_PhjX3latEd6xWKSHAXqXJw"
name="_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parall
el_NegotiateMeetingDate_forever_" type="AND" label="0.2" activity="null"
variables="_PhiwvVatEd6xWKSHAXqXJw" />
  <IAC:Node xmi:id="_PhjX31atEd6xWKSHAXqXJw" name="0.2.1.2.2.3.1" type="START"
label="0.2.1.2.2.3.1" activity="null" />
  <IAC:Node xmi:id="_PhjX4FatEd6xWKSHAXqXJw" name="0.1" type="START" label="0.1"
activity="null" />
  <IAC:Node xmi:id="_PhjX4VatEd6xWKSHAXqXJw" name="0.2.1.2.2.2.5" type="END"
label="0.2.1.2.2.2.5" activity="null" />
```

```
  <IAC:Node xmi:id="_PhjX4latEd6xWKSHAXqXJw" name="ShowResults" type="BASIC"
label="0.2.1.2.2.2.4" activity="send a response to the HMI regarding the user's
request" />
  <IAC:Transition xmi:id="_Phj-0FatEd6xWKSHAXqXJw" TE="null"
source="_PhjX0VatEd6xWKSHAXqXJw" target="_PhjX0VatEd6xWKSHAXqXJw"
name="0.2.2.2.2TO0.2.2.2.2" />
  <IAC:Transition xmi:id="_Phj-0VatEd6xWKSHAXqXJw" TE="null"
source="_PhjXzlatEd6xWKSHAXqXJw" target="_PhjXzFatEd6xWKSHAXqXJw"
name="0.2.2.2TO0.2.2.3" />
  <IAC:Transition xmi:id="_Phj-0latEd6xWKSHAXqXJw" TE="null"
source="_Phiww1atEd6xWKSHAXqXJw" target="_PhiwwFatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.5" />
  <IAC:Transition xmi:id="_Phj-01atEd6xWKSHAXqXJw" TE="null"
source="_PhiwvlatEd6xWKSHAXqXJw" target="_PhiwvFatEd6xWKSHAXqXJw"
name="0.2.2.2.2.1TO0.2.2.2.2.2" />
  <IAC:Transition xmi:id="_Phj-1FatEd6xWKSHAXqXJw" TE="null"
source="_Phiww1atEd6xWKSHAXqXJw" target="_PhiwwVatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.4" />
  <IAC:Transition xmi:id="_Phj-1VatEd6xWKSHAXqXJw" TE="null"
source="_Phiww1atEd6xWKSHAXqXJw" target="_PhiwwlatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.2TO0.2.1.2.2.2.3.3" />
  <IAC:Transition xmi:id="_Phj-1latEd6xWKSHAXqXJw" TE="null"
source="_PhjX21atEd6xWKSHAXqXJw" target="_PhjX2latEd6xWKSHAXqXJw"
name="0.2.1.2.2.3.3TO0.2.1.2.2.3.4" />
  <IAC:Transition xmi:id="_Phj-11atEd6xWKSHAXqXJw" TE="null"
source="_Phiwu1atEd6xWKSHAXqXJw" target="_PhiwuVatEd6xWKSHAXqXJw"
name="0.2.1.2.2.1TO0.2.1.2.2.2" />
  <IAC:Transition xmi:id="_Phj-2FatEd6xWKSHAXqXJw" TE="null"
source="_PhjX3VatEd6xWKSHAXqXJw" target="_PhjX21atEd6xWKSHAXqXJw"
name="0.2.1.2.2.3.2TO0.2.1.2.2.3.3" />
  <IAC:Transition xmi:id="_Phj-2VatEd6xWKSHAXqXJw" TE="null"
source="_PhjX31atEd6xWKSHAXqXJw" target="_PhjX3VatEd6xWKSHAXqXJw"
name="0.2.1.2.2.3.1TO0.2.1.2.2.3.2" />
  <IAC:Transition xmi:id="_Phj-21atEd6xWKSHAXqXJw" TE="accept(p,m,e) or reject(p,m,e)"
source="_PhjXxlatEd6xWKSHAXqXJw" target="_PhjXxVatEd6xWKSHAXqXJw"
name="0.2.2.2.2.3.2.3TO0.2.2.2.2.3.2.4" />
  <IAC:Transition xmi:id="_Phj-21atEd6xWKSHAXqXJw" TE="null"
source="_PhiwuFatEd6xWKSHAXqXJw" target="_PhiwtlatEd6xWKSHAXqXJw"
name="0.2.2.2.2.4TO0.2.2.2.2.5" />
  <IAC:Transition xmi:id="_Phj-3FatEd6xWKSHAXqXJw" TE="null"
source="_Phiws1atEd6xWKSHAXqXJw" target="_Phiws1atEd6xWKSHAXqXJw"
name="0.2.1.2.2TO0.2.1.2.2" />
  <IAC:Transition xmi:id="_Phj-3VatEd6xWKSHAXqXJw" TE="null"
source="_PhiwwFatEd6xWKSHAXqXJw" target="_Phiwv1atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.5TO0.2.1.2.2.2.3.6" />
  <IAC:Transition xmi:id="_Phj-31atEd6xWKSHAXqXJw" TE="null"
source="_PhjX2VatEd6xWKSHAXqXJw" target="_PhjX11atEd6xWKSHAXqXJw"
name="0.2.1.1TO0.2.1.2" />
  <IAC:Transition xmi:id="_Phj-31atEd6xWKSHAXqXJw" TE="null"
source="_PhjXyVatEd6xWKSHAXqXJw" target="_PhjXx1atEd6xWKSHAXqXJw"
name="0.2.2.2.2.3.2.1TO0.2.2.2.2.3.2.2" />
  <IAC:Transition xmi:id="_Phj-4FatEd6xWKSHAXqXJw" TE="null"
source="_PhjX4FatEd6xWKSHAXqXJw" target="_PhjX3latEd6xWKSHAXqXJw" name="0.1TO0.2" />
  <IAC:Transition xmi:id="_Phj-4VatEd6xWKSHAXqXJw" TE="null"
source="_PhjX0latEd6xWKSHAXqXJw" target="_PhjX0VatEd6xWKSHAXqXJw"
name="0.2.2.2.1TO0.2.2.2.2" />
  <IAC:Transition xmi:id="_Phj-41atEd6xWKSHAXqXJw" TE="null"
source="_PhiwwlatEd6xWKSHAXqXJw" target="_Phiwv1atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.3TO0.2.1.2.2.2.3.6" />
  <IAC:Transition xmi:id="_Phj-41atEd6xWKSHAXqXJw" TE="null"
source="_PhjXy1atEd6xWKSHAXqXJw" target="_PhjXylatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.5.3TO0.2.1.2.2.2.3.5.4" />
  <IAC:Transition xmi:id="_Phj-5FatEd6xWKSHAXqXJw" TE="null"
source="_PhiwulatEd6xWKSHAXqXJw" target="_PhiwuFatEd6xWKSHAXqXJw"
name="0.2.2.2.2.3TO0.2.2.2.2.4" />
  <IAC:Transition xmi:id="_Phj-5VatEd6xWKSHAXqXJw" TE="null"
source="_PhjX1FatEd6xWKSHAXqXJw" target="_PhjX01atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.4.3TO0.2.1.2.2.2.3.4.4" />
  <IAC:Transition xmi:id="_Phj-5latEd6xWKSHAXqXJw" TE="null"
source="_PhjX11atEd6xWKSHAXqXJw" target="_PhjX1latEd6xWKSHAXqXJw"
name="0.2.1.2TO0.2.1.3" />
  <IAC:Transition xmi:id="_Phj-51atEd6xWKSHAXqXJw" TE="null"
source="_PhjXzVatEd6xWKSHAXqXJw" target="_PhjXy1atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.5.2TO0.2.1.2.2.2.3.5.3" />
```

```
  <IAC:Transition xmi:id="_Phj-6FatEd6xWKSHAXqXJw" TE="null"
source="_PhiwxFatEd6xWKSHAXqXJw" target="_Phiww1atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.1TO0.2.1.2.2.2.3.2" />
  <IAC:Transition xmi:id="_Phj-6VatEd6xWKSHAXqXJw" TE="null"
source="_Phiwt1atEd6xWKSHAXqXJw" target="_PhiwtVatEd6xWKSHAXqXJw"
name="0.2.1.2.2.3TO0.2.1.2.2.4" />
  <IAC:Transition xmi:id="_Phj-6latEd6xWKSHAXqXJw" TE="null"
source="_PhjX1VatEd6xWKSHAXqXJw" target="_PhjX1FatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.4.2TO0.2.1.2.2.2.3.4.3" />
  <IAC:Transition xmi:id="_Phj-61atEd6xWKSHAXqXJw" TE="propose(m,p,e) or
inform(m,p,e)" source="_PhjXxVatEd6xWKSHAXqXJw" target="_PhjXxFatEd6xWKSHAXqXJw"
name="0.2.2.2.2.3.2.4TO0.2.2.2.2.3.2.5" />
  <IAC:Transition xmi:id="_Phj-7FatEd6xWKSHAXqXJw" TE="null"
source="_PhjXz1atEd6xWKSHAXqXJw" target="_PhjXzVatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.5.1TO0.2.1.2.2.2.3.5.2" />
  <IAC:Transition xmi:id="_Phj-7VatEd6xWKSHAXqXJw" TE="null"
source="_PhiwtFatEd6xWKSHAXqXJw" target="_Phiws1atEd6xWKSHAXqXJw"
name="0.2.1.2.1TO0.2.1.2.2" />
  <IAC:Transition xmi:id="_Phj-7latEd6xWKSHAXqXJw" TE="null"
source="_PhjX4latEd6xWKSHAXqXJw" target="_PhjX4VatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.4TO0.2.1.2.2.2.5" />
  <IAC:Transition xmi:id="_Phkl4FatEd6xWKSHAXqXJw" TE="null"
source="_PhjX2FatEd6xWKSHAXqXJw" target="_PhjX1VatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.4.1TO0.2.1.2.2.2.3.4.2" />
  <IAC:Transition xmi:id="_Phkl4VatEd6xWKSHAXqXJw" TE="null"
source="_PhjX0FatEd6xWKSHAXqXJw" target="_PhjXzlatEd6xWKSHAXqXJw"
name="0.2.2.1TO0.2.2.2" />
  <IAC:Transition xmi:id="_Phkl4latEd6xWKSHAXqXJw" TE="propose(m,p,e)"
source="_PhiwvFatEd6xWKSHAXqXJw" target="_PhiwulatEd6xWKSHAXqXJw"
name="0.2.2.2.2.2TO0.2.2.2.2.3" />
  <IAC:Transition xmi:id="_Phkl41atEd6xWKSHAXqXJw" TE="null"
source="_PhiwsFatEd6xWKSHAXqXJw" target="_PhjX4latEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3TO0.2.1.2.2.2.4" />
  <IAC:Transition xmi:id="_Phkl5FatEd6xWKSHAXqXJw" TE="null"
source="_PhjXx1atEd6xWKSHAXqXJw" target="_PhjXxlatEd6xWKSHAXqXJw"
name="0.2.2.2.2.3.2.2TO0.2.2.2.2.3.2.3" />
  <IAC:Transition xmi:id="_Phkl5VatEd6xWKSHAXqXJw"
TE="inform(m,p,e)/e.isArranged=true" source="_PhjXwFatEd6xWKSHAXqXJw"
target="_PhiwxVatEd6xWKSHAXqXJw" name="0.2.2.2.2.3.2TO0.2.2.2.2.3.3" />
  <IAC:Transition xmi:id="_Phkl51atEd6xWKSHAXqXJw" TE="null"
source="_PhiwsVatEd6xWKSHAXqXJw" target="_PhiwsFatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.2TO0.2.1.2.2.2.3" />
  <IAC:Transition xmi:id="_Phkl51atEd6xWKSHAXqXJw" TE="propose(m,p,e)"
source="_PhjXwFatEd6xWKSHAXqXJw" target="_PhjXwFatEd6xWKSHAXqXJw"
name="0.2.2.2.2.3.2TO0.2.2.2.2.3.2" />
  <IAC:Transition xmi:id="_Phkl6FatEd6xWKSHAXqXJw" TE="null"
source="_PhjXwVatEd6xWKSHAXqXJw" target="_PhjXwFatEd6xWKSHAXqXJw"
name="0.2.2.2.2.3.1TO0.2.2.2.2.3.2" />
  <IAC:Transition xmi:id="_Phkl6VatEd6xWKSHAXqXJw" TE="null"
source="_PhiwslatEd6xWKSHAXqXJw" target="_PhiwsVatEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.1TO0.2.1.2.2.2.2" />
  <IAC:Transition xmi:id="_Phkl6latEd6xWKSHAXqXJw" TE="null"
source="_PhiwuVatEd6xWKSHAXqXJw" target="_Phiwt1atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2TO0.2.1.2.2.3" />
  <IAC:Transition xmi:id="_Phkl61atEd6xWKSHAXqXJw" TE="null"
source="_PhiwwVatEd6xWKSHAXqXJw" target="_Phiwv1atEd6xWKSHAXqXJw"
name="0.2.1.2.2.2.3.4TO0.2.1.2.2.2.3.6" />
  <IAC:Transition xmi:id="_Phkl7FatEd6xWKSHAXqXJw" TE="null"
source="_PhjX3latEd6xWKSHAXqXJw" target="_PhjX3FatEd6xWKSHAXqXJw" name="0.2TO0.3" />
  </xmi:XMI>
```

The automatically generated Java files for the JADE platform

Listing 42. The generated file PersonalAssistantAgent.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;

public class PersonalAssistantAgent extends Agent {

  public void setup() {
    //add behaviour
```

```
    addBehaviour(new
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parallel_Neg
otiateMeetingDate_forever_Behaviour(
        this));
  }
  protected void takeDown() {
    doDelete();
  }
}
```

Listing 43. The generated file _open_group_DecideResponse_sequence_ SendResults_sequence_ReceiveOutcome_close_group__one_or_more_times_ Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.SimpleBehaviour;

public class
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group__o
ne_or_more_times_Behaviour
    extends
      SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder accept = null;
  ACLMessageHolder inform = null;
  ACLMessageHolder propose = null;
  ACLMessageHolder reject = null;
  protected boolean finished;
  Behaviour simpleOneOrMoreTimesBehaviour = null;

  public
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group__o
ne_or_more_times_Behaviour(
      Agent a, MeetingHolder e, ACLMessageHolder accept,
      ACLMessageHolder inform, ACLMessageHolder propose,
      ACLMessageHolder reject) {
    super(a);
    this.e = e;
    this.accept = accept;
    this.inform = inform;
    this.propose = propose;
    this.reject = reject;
    finished = false;
    simpleOneOrMoreTimesBehaviour = new
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group_Be
haviour(
        this.myAgent, e, accept, inform, propose, reject);
    myAgent.addBehaviour(simpleOneOrMoreTimesBehaviour);
  }

  public void action() {
    if (simpleOneOrMoreTimesBehaviour.done()) {
      if (propose(m, p, e)) {
        simpleOneOrMoreTimesBehaviour = new
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group_Be
haviour(
            this.myAgent, e, accept, inform, propose, reject);
        myAgent.addBehaviour(simpleOneOrMoreTimesBehaviour);
      } else
        finished = true;
    }
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 44. The generated file _open_group_DecideResponse_sequence_ SendResults_sequence_ReceiveOutcome_close_group_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group_Be
haviour
    extends
      SequentialBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder accept = null;
  ACLMessageHolder inform = null;
  ACLMessageHolder propose = null;
  ACLMessageHolder reject = null;

  public
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group_Be
haviour(
      Agent a, MeetingHolder e, ACLMessageHolder accept,
      ACLMessageHolder inform, ACLMessageHolder propose,
      ACLMessageHolder reject) {
    super(a);
    this.e = e;
    this.accept = accept;
    this.inform = inform;
    this.propose = propose;
    this.reject = reject;

    addSubBehaviour(new DecideResponseBehaviour(this.myAgent, e, propose));
    addSubBehaviour(new SendResultsBehaviour(this.myAgent, e, accept,
        reject));
    addSubBehaviour(new ReceiveOutcomeBehaviour(this.myAgent, e, inform,
        propose));

  }
}
```

Listing 45. The generated file _open_group_ManageMeetings_sequence_ LearnUserHabits_close_group__forever__parallel_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parallel_Beh
aviour
    extends
      SequentialBehaviour {

  public
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parallel_Beh
aviour(
      Agent a) {
    super(a);

    addSubBehaviour(new
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever_Behaviour(
        this.myAgent));

  }
}
```

Listing 46. The generated file _open_group_ManageMeetings_sequence_ LearnUserHabits_close_group__forever__parallel_NegotiateMeetingDate_forever _Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.ParallelBehaviour;
import jade.core.behaviours.ThreadedBehaviourFactory;

public class
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parallel_Neg
otiateMeetingDate_forever_Behaviour
    extends
      ParallelBehaviour {

  ThreadedBehaviourFactory tbf = null;
  MeetingHolder e = new MeetingHolder(this);

  public
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parallel_Neg
otiateMeetingDate_forever_Behaviour(
      Agent a) {
    super(a, ParallelBehaviour.WHEN_ALL);

    tbf = new ThreadedBehaviourFactory();

    myAgent.addBehaviour(tbf
        .wrap(new NegotiateMeetingDate_forever__parallel_Behaviour(
            this.myAgent, e)));
    myAgent
        .addBehaviour(tbf
            .wrap(new
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever__parallel_Beh
aviour(
                this.myAgent)));
  }
}
```

Listing 47. The generated file _open_group_ManageMeetings_sequence_ LearnUserHabits_close_group__forever_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;

public class
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever_Behaviour
    extends
      CyclicBehaviour {

  Behaviour foreverBehaviour = null;

  public
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group__forever_Behaviour(
      Agent a) {
    super(a);

    foreverBehaviour = new
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group_Behaviour(
        this.myAgent);
    myAgent.addBehaviour(foreverBehaviour);
  }
  public void action() {
    if (foreverBehaviour.done()) {
```

```
      foreverBehaviour = new
_open_group_ManageMeetings_sequence_LearnUserHabits_close_group_Behaviour(
          this.myAgent);
      myAgent.addBehaviour(foreverBehaviour);
    }
  }
}
```

Listing 48. The generated file _open_group_ManageMeetings_sequence_ LearnUserHabits_close_group_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class _open_group_ManageMeetings_sequence_LearnUserHabits_close_group_Behaviour
    extends
      SequentialBehaviour {

  public _open_group_ManageMeetings_sequence_LearnUserHabits_close_group_Behaviour(
      Agent a) {
    super(a);

    addSubBehaviour(new ManageMeetingsBehaviour(this.myAgent));
    addSubBehaviour(new LearnUserHabitsBehaviour(this.myAgent));

  }
}
```

Listing 49. The generated file _open_group_ReadSchedule_or_ RequestChangeMeeting_or_RequestNewMeeting_close_group_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class
_open_group_ReadSchedule_or_RequestChangeMeeting_or_RequestNewMeeting_close_group_Beha
viour
    extends
      SequentialBehaviour {

  public
_open_group_ReadSchedule_or_RequestChangeMeeting_or_RequestNewMeeting_close_group_Beha
viour(Agent a) {
      super(a);

      if (/*insert condition*/)
        addSubBehaviour(new RequestChangeMeetingBehaviour(this.myAgent));
      else if(/*insert condition*/)
        addSubBehaviour(new ReadScheduleBehaviour(this.myAgent));
      else
        addSubBehaviour(new RequestNewMeetingBehaviour(this.myAgent));
    }
}
```

Listing 50. The generated file ACLMessageHolder.java

```
package fr.parisdescartes.mi.meetingsmanagement;
import jade.core.behaviours.Behaviour;
import jade.lang.acl.ACLMessage;

public class ACLMessageHolder {
  ACLMessage aCLMessage = null;
  Behaviour owner;
```

```
  public ACLMessageHolder(Behaviour owner) {
    super();
    this.owner = owner;
  }

  public ACLMessage getACLMessage() {
    return aCLMessage;
  }

  public void setACLMessage(ACLMessage aCLMessage) {
    this.aCLMessage = aCLMessage;
  }

  public Behaviour getOwner() {
    return owner;
  }
}
```

Listing 51. The generated file DecideResponseBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class DecideResponseBehaviour extends SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder propose = null;

  boolean finished = false;

  public DecideResponseBehaviour(Agent a, MeetingHolder e,
      ACLMessageHolder propose) {
    super(a);
    this.e = e;
    this.propose = propose;
  }

  public void action() {

    /*use a reasoning technique to decide if the proposed date matches the user's
profile*/
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 52. The generated file GetUserRequestBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class GetUserRequestBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public GetUserRequestBehaviour(Agent a) {
    super(a);

  }

  public void action() {
```

```
    /*the HMI sends a request*/
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 53. The generated file LearnUserHabitsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class LearnUserHabitsBehaviour extends SequentialBehaviour {

  public LearnUserHabitsBehaviour(Agent a) {
    super(a);

    addSubBehaviour(new LearnUserPreferenceBehaviour(this.myAgent));
    addSubBehaviour(new UpdateUserPreferencesBehaviour(this.myAgent));

  }
}
```

Listing 54. The generated file LearnUserPreferenceBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class LearnUserPreferenceBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public LearnUserPreferenceBehaviour(Agent a) {
    super(a);

  }

  public void action() {

    /*use a simple learning algorithm for the user's preference*/
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 55. The generated file ManageMeetingsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class ManageMeetingsBehaviour extends SequentialBehaviour {

  public ManageMeetingsBehaviour(Agent a) {
    super(a);
```

```
    addSubBehaviour(new GetUserRequestBehaviour(this.myAgent));
    addSubBehaviour(new
_open_group_ReadSchedule_or_RequestChangeMeeting_or_RequestNewMeeting_close_group_Beha
viour(
        this.myAgent));
    addSubBehaviour(new ShowResultsBehaviour(this.myAgent));

  }
}
```

Listing 56. The generated file NegotiateMeetingDate_forever__parallel_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class NegotiateMeetingDate_forever__parallel_Behaviour
    extends
      SequentialBehaviour {

  MeetingHolder e = null;

  public NegotiateMeetingDate_forever__parallel_Behaviour(Agent a,
      MeetingHolder e) {
    super(a);
    this.e = e;

    addSubBehaviour(new NegotiateMeetingDate_forever_Behaviour(
        this.myAgent, e));

  }
}
```

Listing 57. The generated file NegotiateMeetingDate_forever_Behaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;

public class NegotiateMeetingDate_forever_Behaviour extends CyclicBehaviour {

  MeetingHolder e = null;
  Behaviour foreverBehaviour = null;

  public NegotiateMeetingDate_forever_Behaviour(Agent a, MeetingHolder e) {
    super(a);
    this.e = e;
    foreverBehaviour = new NegotiateMeetingDateBehaviour(this.myAgent, e);
    myAgent.addBehaviour(foreverBehaviour);
  }
  public void action() {
    if (foreverBehaviour.done()) {
      foreverBehaviour = new NegotiateMeetingDateBehaviour(this.myAgent,
          e);
      myAgent.addBehaviour(foreverBehaviour);
    }
  }
}
```

Listing 58. The generated file NegotiateMeetingDateBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;
```

```
import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class NegotiateMeetingDateBehaviour extends SequentialBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder accept = new ACLMessageHolder(this);
  ACLMessageHolder inform = new ACLMessageHolder(this);
  ACLMessageHolder propose = new ACLMessageHolder(this);
  ACLMessageHolder reject = new ACLMessageHolder(this);

  public NegotiateMeetingDateBehaviour(Agent a, MeetingHolder e) {
    super(a);
    this.e = e;

    addSubBehaviour(new ReceiveProposedDateBehaviour(this.myAgent, e,
        propose));
    addSubBehaviour(new
_open_group_DecideResponse_sequence_SendResults_sequence_ReceiveOutcome_close_group__o
ne_or_more_times_Behaviour(
        this.myAgent, e, accept, inform, propose, reject));
    addSubBehaviour(new UpdateScheduleBehaviour(this.myAgent, e, inform));

  }
}
```

Listing 59. The generated file ReadScheduleBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class ReadScheduleBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public ReadScheduleBehaviour(Agent a) {
    super(a);

  }

  public void action() {

    /*read the user's schedule from the disk*/
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 60. The generated file ReceiveChangeResultsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class ReceiveChangeResultsBehaviour extends SimpleBehaviour {

  protected MessageTemplate mt = null;
  boolean finished = false;

  public ReceiveChangeResultsBehaviour(Agent a) {
    super(a);
```

```
  }

  public void action() {
    /*insert MessageTemplate code here*/
    ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
      //insert message handling code
      finished = true;
    } else {
      block();
    }
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 61. The generated file ReceiveNewResultsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class ReceiveNewResultsBehaviour extends SimpleBehaviour {

  protected MessageTemplate mt = null;
  boolean finished = false;

  public ReceiveNewResultsBehaviour(Agent a) {
    super(a);

  }

  public void action() {
    /*insert MessageTemplate code here*/
    ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
      //insert message handling code
      finished = true;
    } else {
      block();
    }
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 62. The generated file ReceiveOutcomeBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class ReceiveOutcomeBehaviour extends SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder inform = null;
```

```
  ACLMessageHolder propose = null;
  protected MessageTemplate mt = null;
  boolean finished = false;

  public ReceiveOutcomeBehaviour(Agent a, MeetingHolder e,
      ACLMessageHolder inform, ACLMessageHolder propose) {
    super(a);
    this.e = e;
    this.inform = inform;
    this.propose = propose;
  }

  public void action() {
    mt = MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
    mt = MessageTemplate.or(mt, MessageTemplate
        .MatchPerformative(ACLMessage.INFORM));
    ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
      //insert message handling code
      finished = true;
    } else {
      block();
    }
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 63. The generated file ReceiveProposedDateBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class ReceiveProposedDateBehaviour extends SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder propose = null;
  protected MessageTemplate mt = null;
  boolean finished = false;

  public ReceiveProposedDateBehaviour(Agent a, MeetingHolder e,
      ACLMessageHolder propose) {
    super(a);
    this.e = e;
    this.propose = propose;
  }

  public void action() {
    mt = MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
    ACLMessage msg = myAgent.receive(mt);
    if (msg != null) {
      //insert message handling code
      finished = true;
    } else {
      block();
    }
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 64. The generated file RequestChangeMeetingBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class RequestChangeMeetingBehaviour extends SequentialBehaviour {

  public RequestChangeMeetingBehaviour(Agent a) {
    super(a);

    addSubBehaviour(new SendChangeRequestBehaviour(this.myAgent));
    addSubBehaviour(new ReceiveChangeResultsBehaviour(this.myAgent));

  }
}

RequestNewMeetingBehaviour.java
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class RequestNewMeetingBehaviour extends SequentialBehaviour {

  public RequestNewMeetingBehaviour(Agent a) {
    super(a);

    addSubBehaviour(new SendNewRequestBehaviour(this.myAgent));
    addSubBehaviour(new ReceiveNewResultsBehaviour(this.myAgent));

  }
}
```

Listing 65. The generated file RequestNewMeetingBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SequentialBehaviour;

public class RequestNewMeetingBehaviour extends SequentialBehaviour {

  public RequestNewMeetingBehaviour(Agent a) {
    super(a);

    addSubBehaviour(new SendNewRequestBehaviour(this.myAgent));
    addSubBehaviour(new ReceiveNewResultsBehaviour(this.myAgent));

  }
}
```

Listing 66. The generated file SendChangeRequestBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;

public class SendChangeRequestBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public SendChangeRequestBehaviour(Agent a) {
    super(a);

  }
```

```
  public void action() {
    ACLMessage msg = null;
    //insert message initialization code
    myAgent.send(msg);
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 67. The generated file SendNewRequestBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;

public class SendNewRequestBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public SendNewRequestBehaviour(Agent a) {
    super(a);

  }

  public void action() {
    ACLMessage msg = null;
    //insert message initialization code
    myAgent.send(msg);
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 68. The generated file SendResultsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

import jade.lang.acl.ACLMessage;

public class SendResultsBehaviour extends SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder accept = null;
  ACLMessageHolder reject = null;

  boolean finished = false;

  public SendResultsBehaviour(Agent a, MeetingHolder e,
      ACLMessageHolder accept, ACLMessageHolder reject) {
    super(a);
    this.e = e;
    this.accept = accept;
    this.reject = reject;
  }

  public void action(){
```

```
            ACLMessage msg = null;
    if (/*insert condition*/) {
      msg = new ACLMessage(ACLMessage.ACCEPT);
    }
    else if (/*insert condition*/) {
      msg = new ACLMessage(ACLMessage.REJECT);
    }
    //insert message initialization code
    myAgent.send(msg);
    finished = true;
          }
  public boolean done() {
    return finished;
  }

}
```

Listing 69. The generated file ShowResultsBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class ShowResultsBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public ShowResultsBehaviour(Agent a) {
    super(a);

  }

  public void action() {

    /*send a response to the HMI regarding the user's request*/
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Listing 70. The generated file UpdateScheduleBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class UpdateScheduleBehaviour extends SimpleBehaviour {

  MeetingHolder e = null;
  ACLMessageHolder inform = null;

  boolean finished = false;

  public UpdateScheduleBehaviour(Agent a, MeetingHolder e,
      ACLMessageHolder inform) {
    super(a);
    this.e = e;
    this.inform = inform;
  }

  public void action() {

    /*update the user schedule file on disk*/
    finished = true;
  }
```

```
  public boolean done() {
    return finished;
  }

}
```

Listing 71. The generated file UpdateUserPreferencesBehaviour.java

```
package fr.parisdescartes.mi.meetingsmanagement;

import jade.core.Agent;
import jade.core.behaviours.SimpleBehaviour;

public class UpdateUserPreferencesBehaviour extends SimpleBehaviour {

  boolean finished = false;

  public UpdateUserPreferencesBehaviour(Agent a) {
    super(a);

  }

  public void action() {

    /*update the user preference file on disk*/
    finished = true;
  }

  public boolean done() {
    return finished;
  }

}
```

Annex 7.

Automatically Generated Java Code Using the Rhapsody CASE Tool for the MARKET-MINER Project

```
/*********************************************************************
  Rhapsody : 7.4
  Login    : user
  Component  : DefaultComponent
  Configuration : DefaultConfig
  Model Element  : Product_Pricing_Agent
//! Generated Date: Tue, 14, Apr 2009
  File Path  : DefaultComponent/DefaultConfig/eu/singularlogic/MARKET-
MINER/Product_Pricing_Agent.java
*********************************************************************/

package eu.singularlogic.MARKET-MINER;

//## auto_generated
import java.util.*;
//## auto_generated
import com.telelogic.rhapsody.oxf.*;
//## auto_generated
import com.telelogic.rhapsody.animcom.*;
//## auto_generated
import com.telelogic.rhapsody.oxfinst.*;
//## auto_generated
import com.telelogic.rhapsody.animation.*;
//## auto_generated
import com.telelogic.rhapsody.oxf.states.*;
//## auto_generated
import com.telelogic.rhapsody.oxf.timeouts.*;
//## auto_generated
import com.telelogic.rhapsody.animcom.animMessages.*;
//## attribute firmStrategy
import eu.singularlogic.MARKET-MINER.ontology.FirmStrategy;

//-------------------------------------------------------------------------
// eu/singularlogic/MARKET-MINER/Product_Pricing_Agent.java
//-------------------------------------------------------------------------
```

```
//## package eu::singularlogic::MARKET-MINER


//## class Product_Pricing_Agent
public class Product_Pricing_Agent implements RiJStateConcept, Animated {

    public Reactive reactive;   //## ignore

    protected FirmStrategy firmStrategy = null;   //## attribute firmStrategy

    protected int pricingInterval = 86400000;   //## attribute pricingInterval

    protected int productTypes = null;   //## attribute productTypes

    protected java.util.Hashtable products = null;   //## attribute products

    protected boolean userClosedGUI = false;  //## attribute userClosedGUI

    //#[ ignore
    public static final int RiJNonState=0;
    public static final int ProductPricingAgent=1;
    public static final int OptionalForeverGetMarketInformation=2;
    public static final int state_23=3;
    public static final int ForeverGetMarketInformation=4;
    public static final int GetMarketInformation=5;
    public static final int UpdateFacts=6;
    public static final int state_28=7;
    public static final int GetWeatherInformation=8;
    public static final int GetLocalInformation=9;
    public static final int GetCompetitionInformation=10;
    public static final int ForeverInteractWithUser=11;
    public static final int InteractWithUser=12;
    public static final int state_18=13;
    public static final int PresentInformationToTheUserOrUpdateFirmPolicy=14;
    public static final int UpdateFirmPolicy=15;
    public static final int state_21=16;
    public static final int PresentInformationToTheUser=17;
    public static final int ForeverDecideOnPricingPolicy=18;
    public static final int DecideOnPricingPolicy=19;
    public static final int WaitForNewPeriod=20;
    public static final int state_15=21;
    public static final int GetProductsInformation=22;
    public static final int FixPrices=23;
    public static final int DeterminePricingPolicy=24;
    //#]
    protected int rootState_subState;  //## ignore

    protected int rootState_active;  //## ignore

    protected int OptionalForeverGetMarketInformation_subState;   //## ignore

    protected int OptionalForeverGetMarketInformation_active;   //## ignore

    protected int ForeverGetMarketInformation_subState;    //## ignore

    protected int GetMarketInformation_subState;    //## ignore

    protected int ForeverInteractWithUser_subState;  //## ignore

    protected int ForeverInteractWithUser_active;  //## ignore

    protected int InteractWithUser_subState;  //## ignore

    protected int PresentInformationToTheUserOrUpdateFirmPolicy_subState;    //##
ignore

    protected int ForeverDecideOnPricingPolicy_subState;   //## ignore

    protected int ForeverDecideOnPricingPolicy_active;   //## ignore

    protected int DecideOnPricingPolicy_subState;   //## ignore

    //#[ ignore
    // Instrumentation attributes (Animation)
    private Animate animate;
```

```
    public static AnimClass animClassProduct_Pricing_Agent = new
AnimClass("eu.singularlogic.MARKET-MINER.Product_Pricing_Agent",false);
    //#]


    //## statechart_method
    public RiJThread getThread() {
        return reactive.getThread();
    }

    //## statechart_method
    public void schedTimeout(long delay, long tmID, RiJStateReactive reactive) {
        getThread().schedTimeout(delay, tmID, reactive);
    }

    //## statechart_method
    public void unschedTimeout(long tmID, RiJStateReactive reactive) {
        getThread().unschedTimeout(tmID, reactive);
    }

    //## statechart_method
    public boolean isIn(int state) {
        return reactive.isIn(state);
    }

    //## statechart_method
    public boolean isCompleted(int state) {
        return reactive.isCompleted(state);
    }

    //## statechart_method
    public RiJEventConsumer getEventConsumer() {
        return (RiJEventConsumer)reactive;
    }

    //## statechart_method
    public void gen(RiJEvent event) {
        reactive._gen(event);
    }

    //## statechart_method
    public void queueEvent(RiJEvent event) {
        reactive.queueEvent(event);
    }

    //## statechart_method
    public int takeEvent(RiJEvent event) {
        return reactive.takeEvent(event);
    }

    // Constructors

    //## auto_generated
    public  Product_Pricing_Agent(RiJThread p_thread) {
        try {

animInstance().notifyConstructorEntered(animClassProduct_Pricing_Agent.getUserClass(),
                new ArgData[] {
                });

        reactive = new Reactive(p_thread);
        }
        finally {
            animInstance().notifyMethodExit();
        }

    }

    //## auto_generated
    public FirmStrategy getFirmStrategy() {
        return firmStrategy;
    }

    //## auto_generated
    public void setFirmStrategy(FirmStrategy p_firmStrategy) {
```

```
        firmStrategy = p_firmStrategy;
    }

    //## auto_generated
    public int getPricingInterval() {
        return pricingInterval;
    }

    //## auto_generated
    public void setPricingInterval(int p_pricingInterval) {
        pricingInterval = p_pricingInterval;
    }

    //## auto_generated
    public int getProductTypes() {
        return productTypes;
    }

    //## auto_generated
    public void setProductTypes(int p_productTypes) {
        productTypes = p_productTypes;
    }

    //## auto_generated
    public java.util.Hashtable getProducts() {
        return products;
    }

    //## auto_generated
    public void setProducts(java.util.Hashtable p_products) {
        products = p_products;
    }

    //## auto_generated
    public boolean getUserClosedGUI() {
        return userClosedGUI;
    }

    //## auto_generated
    public void setUserClosedGUI(boolean p_userClosedGUI) {
        userClosedGUI = p_userClosedGUI;
    }

    //## auto_generated
    public boolean startBehavior() {
        boolean done = false;
        done = reactive.startBehavior();
        return done;
    }

    //## ignore
    public class Reactive extends RiJStateReactive implements AnimatedReactive {

        // Default constructor
        public Reactive() {
            this(RiJMainThread.instance());
        }


        // Constructors

        public  Reactive(RiJThread p_thread) {
            super(p_thread);
            initStatechart();
        }

        //## statechart_method
        public boolean isIn(int state) {
            if(DecideOnPricingPolicy_subState == state)
                {
                    return true;
                }
            if(ForeverDecideOnPricingPolicy == state)
                {
                    return isIn(ProductPricingAgent);
                }
```

```
            if(ForeverDecideOnPricingPolicy_subState == state)
                {
                    return true;
                }
            if(PresentInformationToTheUserOrUpdateFirmPolicy_subState == state)
                {
                    return true;
                }
            if(InteractWithUser_subState == state)
                {
                    return true;
                }
            if(ForeverInteractWithUser == state)
                {
                    return isIn(ProductPricingAgent);
                }
            if(ForeverInteractWithUser_subState == state)
                {
                    return true;
                }
            if(GetMarketInformation_subState == state)
                {
                    return true;
                }
            if(ForeverGetMarketInformation_subState == state)
                {
                    return true;
                }
            if(OptionalForeverGetMarketInformation == state)
                {
                    return isIn(ProductPricingAgent);
                }
            if(OptionalForeverGetMarketInformation_subState == state)
                {
                    return true;
                }
            if(rootState_subState == state)
                {
                    return true;
                }
            return false;
        }

        //## statechart_method
        public boolean isCompleted(int state) {
            if(DecideOnPricingPolicy == state)
                {
                    return ( isIn(state_15) );
                }
            if(PresentInformationToTheUserOrUpdateFirmPolicy == state)
                {
                    return ( isIn(state_21) );
                }
            if(InteractWithUser == state)
                {
                    return ( isIn(state_18) );
                }
            if(GetMarketInformation == state)
                {
                    return ( isIn(state_28) );
                }
            if(OptionalForeverGetMarketInformation == state)
                {
                    return ( isIn(state_23) );
                }
            if(ProductPricingAgent == state)
                {
                    if(isCompleted(OptionalForeverGetMarketInformation) == false)
                        {
                            return false;
                        }
                    return true;
                }
            return true;
        }
```

```
        //## statechart_method
        public void rootState_add(AnimStates animStates) {
            animStates.add("ROOT");
            if(rootState_subState == ProductPricingAgent)
                {
                    ProductPricingAgent_add(animStates);
                }
        }

        //## statechart_method
        public void rootState_entDef() {
            {
                rootState_enter();
                rootStateEntDef();
            }
        }

        //## statechart_method
        public int rootState_dispatchEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(rootState_active == ProductPricingAgent)
                {
                    res = ProductPricingAgent_dispatchEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void ProductPricingAgent_add(AnimStates animStates) {
            animStates.add("ROOT.ProductPricingAgent");
            OptionalForeverGetMarketInformation_add(animStates);
            ForeverDecideOnPricingPolicy_add(animStates);
            ForeverInteractWithUser_add(animStates);
        }

        //## statechart_method
        public int ProductPricingAgent_dispatchEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(OptionalForeverGetMarketInformation_dispatchEvent(id) >= 0)
                {
                    res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                    if(!isIn(ProductPricingAgent))
                        {
                            return res;
                        }
                }
            if(ForeverDecideOnPricingPolicy_dispatchEvent(id) >= 0)
                {
                    res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                    if(!isIn(ProductPricingAgent))
                        {
                            return res;
                        }
                }
            if(ForeverInteractWithUser_dispatchEvent(id) >= 0)
                {
                    res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                    if(!isIn(ProductPricingAgent))
                        {
                            return res;
                        }
                }
            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = ProductPricingAgent_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void OptionalForeverGetMarketInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation");
            switch (OptionalForeverGetMarketInformation_subState) {
                case ForeverGetMarketInformation:
                {
```

```
                    ForeverGetMarketInformation_add(animStates);
                    break;
                }
                case state_23:
                {
                    state_23_add(animStates);
                    break;
                }
                default:
                    break;
            }
        }

        //## statechart_method
        public int OptionalForeverGetMarketInformation_dispatchEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            switch (OptionalForeverGetMarketInformation_active) {
                case GetWeatherInformation:
                {
                    res = GetWeatherInformation_takeEvent(id);
                    break;
                }
                case GetLocalInformation:
                {
                    res = GetLocalInformation_takeEvent(id);
                    break;
                }
                case GetCompetitionInformation:
                {
                    res = GetCompetitionInformation_takeEvent(id);
                    break;
                }
                case state_28:
                {
                    res = state_28_takeEvent(id);
                    break;
                }
                case UpdateFacts:
                {
                    res = UpdateFacts_takeEvent(id);
                    break;
                }
                case state_23:
                {
                    res = state_23_takeEvent(id);
                    break;
                }
                default:
                    break;
            }
            return res;
        }

        //## statechart_method
        public void state_23_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.state_23"
);
        }

        //## statechart_method
        public void ForeverGetMarketInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation");
            if(ForeverGetMarketInformation_subState == GetMarketInformation)
                {
                    GetMarketInformation_add(animStates);
                }
        }

        //## statechart_method
        public void GetMarketInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation.GetMarketInformation");
```

```
            switch (GetMarketInformation_subState) {
                case GetWeatherInformation:
                {
                    GetWeatherInformation_add(animStates);
                    break;
                }
                case GetLocalInformation:
                {
                    GetLocalInformation_add(animStates);
                    break;
                }
                case GetCompetitionInformation:
                {
                    GetCompetitionInformation_add(animStates);
                    break;
                }
                case state_28:
                {
                    state_28_add(animStates);
                    break;
                }
                case UpdateFacts:
                {
                    UpdateFacts_add(animStates);
                    break;
                }
                default:
                    break;
            }
        }

        //## statechart_method
        public void UpdateFacts_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation.GetMarketInformation.UpdateFacts");
        }

        //## statechart_method
        public void state_28_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation.GetMarketInformation.state_28");
        }

        //## statechart_method
        public void GetWeatherInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation.GetMarketInformation.GetWeatherInformation");
        }

        //## statechart_method
        public void GetLocalInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation.GetMarketInformation.GetLocalInformation");
        }

        //## statechart_method
        public void GetCompetitionInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.OptionalForeverGetMarketInformation.ForeverGe
tMarketInformation.GetMarketInformation.GetCompetitionInformation");
        }

        //## statechart_method
        public void ForeverInteractWithUser_add(AnimStates animStates) {
            animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser");
            if(ForeverInteractWithUser_subState == InteractWithUser)
                {
                    InteractWithUser_add(animStates);
                }
        }

        //## statechart_method
```

```
        public int ForeverInteractWithUser_dispatchEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            switch (ForeverInteractWithUser_active) {
                case PresentInformationToTheUser:
                {
                    res = PresentInformationToTheUser_takeEvent(id);
                    break;
                }
                case UpdateFirmPolicy:
                {
                    res = UpdateFirmPolicy_takeEvent(id);
                    break;
                }
                case state_21:
                {
                    res = state_21_takeEvent(id);
                    break;
                }
                case state_18:
                {
                    res = state_18_takeEvent(id);
                    break;
                }
                default:
                    break;
            }
            return res;
        }

        //## statechart_method
        public void InteractWithUser_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser.InteractWithUser");
            switch (InteractWithUser_subState) {
                case PresentInformationToTheUserOrUpdateFirmPolicy:
                {
                    PresentInformationToTheUserOrUpdateFirmPolicy_add(animStates);
                    break;
                }
                case state_18:
                {
                    state_18_add(animStates);
                    break;
                }
                default:
                    break;
            }
        }

        //## statechart_method
        public void state_18_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser.InteractWithUser.stat
e_18");
        }

        //## statechart_method
        public void PresentInformationToTheUserOrUpdateFirmPolicy_add(AnimStates
animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser.InteractWithUser.Pres
entInformationToTheUserOrUpdateFirmPolicy");
            switch (PresentInformationToTheUserOrUpdateFirmPolicy_subState) {
                case PresentInformationToTheUser:
                {
                    PresentInformationToTheUser_add(animStates);
                    break;
                }
                case UpdateFirmPolicy:
                {
                    UpdateFirmPolicy_add(animStates);
                    break;
                }
                case state_21:
                {
                    state_21_add(animStates);
```

```
                    break;
                }
                default:
                    break;
            }
        }

        //## statechart_method
        public void UpdateFirmPolicy_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser.InteractWithUser.Pres
entInformationToTheUserOrUpdateFirmPolicy.UpdateFirmPolicy");
        }

        //## statechart_method
        public void state_21_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser.InteractWithUser.Pres
entInformationToTheUserOrUpdateFirmPolicy.state_21");
        }

        //## statechart_method
        public void PresentInformationToTheUser_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverInteractWithUser.InteractWithUser.Pres
entInformationToTheUserOrUpdateFirmPolicy.PresentInformationToTheUser");
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicy_add(AnimStates animStates) {
            animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy");
            if(ForeverDecideOnPricingPolicy_subState == DecideOnPricingPolicy)
                {
                    DecideOnPricingPolicy_add(animStates);
                }
        }

        //## statechart_method
        public int ForeverDecideOnPricingPolicy_dispatchEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            switch (ForeverDecideOnPricingPolicy_active) {
                case WaitForNewPeriod:
                {
                    res = WaitForNewPeriod_takeEvent(id);
                    break;
                }
                case GetProductsInformation:
                {
                    res = GetProductsInformation_takeEvent(id);
                    break;
                }
                case DeterminePricingPolicy:
                {
                    res = DeterminePricingPolicy_takeEvent(id);
                    break;
                }
                case FixPrices:
                {
                    res = FixPrices_takeEvent(id);
                    break;
                }
                case state_15:
                {
                    res = state_15_takeEvent(id);
                    break;
                }
                default:
                    break;
            }
            return res;
        }

        //## statechart_method
        public void DecideOnPricingPolicy_add(AnimStates animStates) {
```

```
animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingP
olicy");
            switch (DecideOnPricingPolicy_subState) {
                case WaitForNewPeriod:
                {
                    WaitForNewPeriod_add(animStates);
                    break;
                }
                case GetProductsInformation:
                {
                    GetProductsInformation_add(animStates);
                    break;
                }
                case DeterminePricingPolicy:
                {
                    DeterminePricingPolicy_add(animStates);
                    break;
                }
                case FixPrices:
                {
                    FixPrices_add(animStates);
                    break;
                }
                case state_15:
                {
                    state_15_add(animStates);
                    break;
                }
                default:
                    break;
            }
        }

        //## statechart_method
        public void WaitForNewPeriod_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingP
olicy.WaitForNewPeriod");
        }

        //## statechart_method
        public void state_15_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingP
olicy.state_15");
        }

        //## statechart_method
        public void GetProductsInformation_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingP
olicy.GetProductsInformation");
        }

        //## statechart_method
        public void FixPrices_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingP
olicy.FixPrices");
        }

        //## statechart_method
        public void DeterminePricingPolicy_add(AnimStates animStates) {

animStates.add("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingP
olicy.DeterminePricingPolicy");
        }

        //## auto_generated
        protected void initStatechart() {
            rootState_subState = RiJNonState;
            rootState_active = RiJNonState;
            OptionalForeverGetMarketInformation_subState = RiJNonState;
            OptionalForeverGetMarketInformation_active = RiJNonState;
            ForeverGetMarketInformation_subState = RiJNonState;
```

```
            GetMarketInformation_subState = RiJNonState;
            ForeverInteractWithUser_subState = RiJNonState;
            ForeverInteractWithUser_active = RiJNonState;
            InteractWithUser_subState = RiJNonState;
            PresentInformationToTheUserOrUpdateFirmPolicy_subState = RiJNonState;
            ForeverDecideOnPricingPolicy_subState = RiJNonState;
            ForeverDecideOnPricingPolicy_active = RiJNonState;
            DecideOnPricingPolicy_subState = RiJNonState;
        }

        //## statechart_method
        public int FixPricesTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("6");
            FixPrices_exit();
            state_15_entDef();
            animInstance().notifyTransitionEnded("6");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void WaitForNewPeriod_entDef() {
            WaitForNewPeriod_enter();
        }

        //## statechart_method
        public void DecideOnPricingPolicyEnter() {
        }

        //## statechart_method
        public void ForeverInteractWithUser_exit() {
            if(ForeverInteractWithUser_subState == InteractWithUser)
                {
                    InteractWithUser_exit();
                }
            ForeverInteractWithUser_subState = RiJNonState;
            ForeverInteractWithUserExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser");
        }

        //## statechart_method
        public void GetCompetitionInformationEnter() {
        }

        //## statechart_method
        public void GetLocalInformation_entDef() {
            GetLocalInformation_enter();
        }

        //## statechart_method
        public void GetProductsInformationEnter() {
            //#[ state
ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingPolicy.GetProduct
sInformation.(Entry)
            //connect to web service
            //#]
        }

        //## statechart_method
        public int PresentInformationToTheUserOrUpdateFirmPolicyTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(isCompleted(PresentInformationToTheUserOrUpdateFirmPolicy))
                {
                    //## transition 10
                    if(userClosedGUI==true)
                        {
                            animInstance().notifyTransitionStarted("10");
                            PresentInformationToTheUserOrUpdateFirmPolicy_exit();
                            state_18_entDef();
                            animInstance().notifyTransitionEnded("10");
                            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                        }
                    else
                        {
```

```
                            //## transition 11
                            if(userClosedGUI==false)
                                {
                                    animInstance().notifyTransitionStarted("11");

PresentInformationToTheUserOrUpdateFirmPolicy_exit();

PresentInformationToTheUserOrUpdateFirmPolicy_entDef();
                                    animInstance().notifyTransitionEnded("11");
                                    res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                                }
                        }
                }
            return res;
        }

        //## statechart_method
        public int InteractWithUserTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            //## transition 8
            if(isCompleted(InteractWithUser))
                {
                    animInstance().notifyTransitionStarted("8");
                    InteractWithUser_exit();
                    InteractWithUser_entDef();
                    animInstance().notifyTransitionEnded("8");
                    res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                }
            return res;
        }

        //## statechart_method
        public void ForeverGetMarketInformation_exit() {
            popNullConfig();
            if(ForeverGetMarketInformation_subState == GetMarketInformation)
                {
                    GetMarketInformation_exit();
                }
            ForeverGetMarketInformation_subState = RiJNonState;
            ForeverGetMarketInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation");
        }

        //## statechart_method
        public void OptionalForeverGetMarketInformationExit() {
        }

        //## statechart_method
        public void DeterminePricingPolicyExit() {
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicy_exit() {
            if(ForeverDecideOnPricingPolicy_subState == DecideOnPricingPolicy)
                {
                    DecideOnPricingPolicy_exit();
                }
            ForeverDecideOnPricingPolicy_subState = RiJNonState;
            ForeverDecideOnPricingPolicyExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y");
        }

        //## statechart_method
        public void state_21_entDef() {
            state_21_enter();
        }

        //## statechart_method
        public void state_18Enter() {
        }

        //## statechart_method
```

```
        public int GetWeatherInformationTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("23");
            GetWeatherInformation_exit();
            GetLocalInformation_entDef();
            animInstance().notifyTransitionEnded("23");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void UpdateFacts_entDef() {
            UpdateFacts_enter();
        }

        //## statechart_method
        public int state_23_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            res = OptionalForeverGetMarketInformation_takeEvent(id);
            return res;
        }

        //## statechart_method
        public void ProductPricingAgentEnter() {
        }

        //## statechart_method
        public void DeterminePricingPolicyEnter() {
        }

        //## statechart_method
        public int state_15_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            res = DecideOnPricingPolicy_takeEvent(id);
            return res;
        }

        //## statechart_method
        public int WaitForNewPeriod_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = WaitForNewPeriodTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = DecideOnPricingPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void DecideOnPricingPolicy_exit() {
            popNullConfig();
            switch (DecideOnPricingPolicy_subState) {
                case WaitForNewPeriod:
                {
                    WaitForNewPeriod_exit();
                    break;
                }
                case GetProductsInformation:
                {
                    GetProductsInformation_exit();
                    break;
                }
                case DeterminePricingPolicy:
                {
                    DeterminePricingPolicy_exit();
                    break;
                }
                case FixPrices:
                {
                    FixPrices_exit();
                    break;
                }
```

```
                case state_15:
                {
                    state_15_exit();
                    break;
                }
                default:
                    break;
            }
            DecideOnPricingPolicy_subState = RiJNonState;
            DecideOnPricingPolicyExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y.DecideOnPricingPolicy");
        }

        //## statechart_method
        public void DecideOnPricingPolicy_entDef() {
            DecideOnPricingPolicy_enter();

            animInstance().notifyTransitionStarted("2");
            WaitForNewPeriod_entDef();
            animInstance().notifyTransitionEnded("2");
        }

        //## statechart_method
        public void PresentInformationToTheUserEnter() {
        }

        //## statechart_method
        public int state_18_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            res = InteractWithUser_takeEvent(id);
            return res;
        }

        //## statechart_method
        public void GetCompetitionInformation_exit() {
            popNullConfig();
            GetCompetitionInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation.GetMarketInformation.GetCompetitionInformation");
        }

        //## statechart_method
        public int GetLocalInformationTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("24");
            GetLocalInformation_exit();
            GetCompetitionInformation_entDef();
            animInstance().notifyTransitionEnded("24");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void state_28Enter() {
        }

        //## statechart_method
        public void UpdateFactsEnter() {
        }

        //## statechart_method
        public int ForeverGetMarketInformationTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("20");
            ForeverGetMarketInformation_exit();
            state_23_entDef();
            animInstance().notifyTransitionEnded("20");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public int ProductPricingAgent_takeEvent(short id) {
```

```
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            return res;
        }

        //## statechart_method
        public void GetProductsInformation_exit() {
            popNullConfig();
            GetProductsInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y.DecideOnPricingPolicy.GetProductsInformation");
        }

        //## statechart_method
        public void state_15_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy.DecideOnPricingPolicy.state_15");
            DecideOnPricingPolicy_subState = state_15;
            ForeverDecideOnPricingPolicy_active = state_15;
            state_15Enter();
        }

        //## statechart_method
        public void DecideOnPricingPolicy_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy.DecideOnPricingPolicy");
            pushNullConfig();
            ForeverDecideOnPricingPolicy_subState = DecideOnPricingPolicy;
            DecideOnPricingPolicyEnter();
        }

        //## statechart_method
        public void UpdateFirmPolicyExit() {
        }

        //## statechart_method
        public int GetWeatherInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = GetWeatherInformationTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = GetMarketInformation_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void FixPricesEnter() {
        }

        //## statechart_method
        public void state_15Enter() {
        }

        //## statechart_method
        public void WaitForNewPeriodEnter() {
            //#[ state
ROOT.ProductPricingAgent.ForeverDecideOnPricingPolicy.DecideOnPricingPolicy.WaitForNew
Period.(Entry)
            //Wait for a time equal to the TIME_PERIOD variable
            //#]
        }

        //## statechart_method
        public void DecideOnPricingPolicyExit() {
        }

        //## statechart_method
        public int UpdateFirmPolicy_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
```

```
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = UpdateFirmPolicyTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = PresentInformationToTheUserOrUpdateFirmPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void state_18_exit() {
            state_18Exit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser.Int
eractWithUser.state_18");
        }

        //## statechart_method
        public void GetLocalInformationExit() {
        }

        //## statechart_method
        public void GetLocalInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation.GetMarketInformation.GetLocalInformation");
            pushNullConfig();
            GetMarketInformation_subState = GetLocalInformation;
            OptionalForeverGetMarketInformation_active = GetLocalInformation;
            GetLocalInformationEnter();
        }

        //## statechart_method
        public void GetMarketInformation_entDef() {
            GetMarketInformation_enter();

            animInstance().notifyTransitionStarted("22");
            GetWeatherInformation_entDef();
            animInstance().notifyTransitionEnded("22");
        }

        //## statechart_method
        public int OptionalForeverGetMarketInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            return res;
        }

        //## statechart_method
        public void ProductPricingAgent_exit() {
            OptionalForeverGetMarketInformation_exit();
            ForeverDecideOnPricingPolicy_exit();
            ForeverInteractWithUser_exit();
            ProductPricingAgentExit();
            animInstance().notifyStateExited("ROOT.ProductPricingAgent");
        }

        //## statechart_method
        public void DeterminePricingPolicy_exit() {
            popNullConfig();
            DeterminePricingPolicyExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y.DecideOnPricingPolicy.DeterminePricingPolicy");
        }

        //## statechart_method
        public int FixPrices_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = FixPricesTakeNull();
                }
```

```
            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = DecideOnPricingPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void WaitForNewPeriodExit() {
        }

        //## statechart_method
        public void PresentInformationToTheUser_exit() {
            popNullConfig();
            PresentInformationToTheUserExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser.Int
eractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy.PresentInformationToTheUse
r");
        }

        //## statechart_method
        public void state_21Exit() {
        }

        //## statechart_method
        public void state_18_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser.In
teractWithUser.state_18");
            InteractWithUser_subState = state_18;
            ForeverInteractWithUser_active = state_18;
            state_18Enter();
        }

        //## statechart_method
        public void state_28_exit() {
            state_28Exit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation.GetMarketInformation.state_28");
        }

        //## statechart_method
        public int UpdateFacts_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = UpdateFactsTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = GetMarketInformation_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void UpdateFacts_exit() {
            popNullConfig();
            UpdateFactsExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation.GetMarketInformation.UpdateFacts");
        }

        //## statechart_method
        public void UpdateFacts_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation.GetMarketInformation.UpdateFacts");
            pushNullConfig();
            GetMarketInformation_subState = UpdateFacts;
            OptionalForeverGetMarketInformation_active = UpdateFacts;
            UpdateFactsEnter();
```

```
        }

        //## statechart_method
        public void GetMarketInformationExit() {
        }

        //## statechart_method
        public void GetMarketInformationEnter() {
        }

        //## statechart_method
        public int rootState_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            return res;
        }

        //## statechart_method
        public void PresentInformationToTheUserExit() {
        }

        //## statechart_method
        public void PresentInformationToTheUser_entDef() {
            PresentInformationToTheUser_enter();
        }

        //## statechart_method
        public void state_21_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser.In
teractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy.state_21");
            PresentInformationToTheUserOrUpdateFirmPolicy_subState = state_21;
            ForeverInteractWithUser_active = state_21;
            state_21Enter();
        }

        //## statechart_method
        public void UpdateFirmPolicy_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser.In
teractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy.UpdateFirmPolicy");
            pushNullConfig();
            PresentInformationToTheUserOrUpdateFirmPolicy_subState = UpdateFirmPolicy;
            ForeverInteractWithUser_active = UpdateFirmPolicy;
            UpdateFirmPolicyEnter();
        }

        //## statechart_method
        public void ForeverInteractWithUserExit() {
        }

        //## statechart_method
        public void ForeverInteractWithUser_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser");
            ForeverInteractWithUserEnter();
        }

        //## statechart_method
        public void GetMarketInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation.GetMarketInformation");
            ForeverGetMarketInformation_subState = GetMarketInformation;
            GetMarketInformationEnter();
        }

        //## statechart_method
        public void ForeverGetMarketInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation");
            pushNullConfig();
            OptionalForeverGetMarketInformation_subState =
ForeverGetMarketInformation;
            ForeverGetMarketInformationEnter();
        }
```

```
        //## statechart_method
        public void OptionalForeverGetMarketInformationEnter() {
        }

        //## statechart_method
        public void ProductPricingAgent_entDef() {
            ProductPricingAgent_enter();
            OptionalForeverGetMarketInformation_entDef();
            ForeverDecideOnPricingPolicy_entDef();
            ForeverInteractWithUser_entDef();
        }

        //## statechart_method
        public void DeterminePricingPolicy_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy.DecideOnPricingPolicy.DeterminePricingPolicy");
            pushNullConfig();
            DecideOnPricingPolicy_subState = DeterminePricingPolicy;
            ForeverDecideOnPricingPolicy_active = DeterminePricingPolicy;
            DeterminePricingPolicyEnter();
        }

        //## statechart_method
        public void FixPrices_exit() {
            popNullConfig();
            FixPricesExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y.DecideOnPricingPolicy.FixPrices");
        }

        //## statechart_method
        public void FixPricesExit() {
        }

        //## statechart_method
        public void GetProductsInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy.DecideOnPricingPolicy.GetProductsInformation");
            pushNullConfig();
            DecideOnPricingPolicy_subState = GetProductsInformation;
            ForeverDecideOnPricingPolicy_active = GetProductsInformation;
            GetProductsInformationEnter();
        }

        //## statechart_method
        public void GetProductsInformation_entDef() {
            GetProductsInformation_enter();
        }

        //## statechart_method
        public void state_15_exit() {
            state_15Exit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y.DecideOnPricingPolicy.state_15");
        }

        //## statechart_method
        public void WaitForNewPeriod_exit() {
            popNullConfig();
            WaitForNewPeriodExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverDecideOnPricingPolic
y.DecideOnPricingPolicy.WaitForNewPeriod");
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicy_entDef() {
            ForeverDecideOnPricingPolicy_enter();
            ForeverDecideOnPricingPolicyEntDef();
        }
```

```
        //## statechart_method
        public void InteractWithUser_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser.In
teractWithUser");
            pushNullConfig();
            ForeverInteractWithUser_subState = InteractWithUser;
            InteractWithUserEnter();
        }

        //## statechart_method
        public void GetWeatherInformationEnter() {
        }

        //## statechart_method
        public int state_28_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            res = GetMarketInformation_takeEvent(id);
            return res;
        }

        //## statechart_method
        public int UpdateFactsTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("26");
            UpdateFacts_exit();
            state_28_entDef();
            animInstance().notifyTransitionEnded("26");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void ForeverGetMarketInformationExit() {
        }

        //## statechart_method
        public void state_23Exit() {
        }

        //## statechart_method
        public void state_23Enter() {
        }

        //## statechart_method
        public void rootState_enter() {
            animInstance().notifyStateEntered("ROOT");
            rootStateEnter();
        }

        //## statechart_method
        public void rootStateEnter() {
        }

        //## statechart_method
        public int DeterminePricingPolicyTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("5");
            DeterminePricingPolicy_exit();
            FixPrices_entDef();
            animInstance().notifyTransitionEnded("5");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void FixPrices_entDef() {
            FixPrices_enter();
        }

        //## statechart_method
        public void WaitForNewPeriod_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy.DecideOnPricingPolicy.WaitForNewPeriod");
            pushNullConfig();
```

```
            DecideOnPricingPolicy_subState = WaitForNewPeriod;
            ForeverDecideOnPricingPolicy_active = WaitForNewPeriod;
            WaitForNewPeriodEnter();
        }

        //## statechart_method
        public int ForeverDecideOnPricingPolicy_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            return res;
        }

        //## statechart_method
        public void UpdateFirmPolicyEnter() {
        }

        //## statechart_method
        public int GetCompetitionInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = GetCompetitionInformationTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = GetMarketInformation_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public int GetLocalInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = GetLocalInformationTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = GetMarketInformation_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void GetMarketInformation_exit() {
            switch (GetMarketInformation_subState) {
                case GetWeatherInformation:
                {
                    GetWeatherInformation_exit();
                    break;
                }
                case GetLocalInformation:
                {
                    GetLocalInformation_exit();
                    break;
                }
                case GetCompetitionInformation:
                {
                    GetCompetitionInformation_exit();
                    break;
                }
                case state_28:
                {
                    state_28_exit();
                    break;
                }
                case UpdateFacts:
                {
                    UpdateFacts_exit();
                    break;
                }
                default:
                    break;
            }
```

```
            GetMarketInformation_subState = RiJNonState;
            GetMarketInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation.GetMarketInformation");
        }

        //## statechart_method
        public void state_23_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.state_23");
            OptionalForeverGetMarketInformation_subState = state_23;
            OptionalForeverGetMarketInformation_active = state_23;
            state_23Enter();
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicy_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy");
            ForeverDecideOnPricingPolicyEnter();
        }

        //## statechart_method
        public void state_21Enter() {
        }

        //## statechart_method
        public int PresentInformationToTheUserOrUpdateFirmPolicy_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = PresentInformationToTheUserOrUpdateFirmPolicyTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = InteractWithUser_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void PresentInformationToTheUserOrUpdateFirmPolicy_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser.In
teractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy");
            pushNullConfig();
            InteractWithUser_subState = PresentInformationToTheUserOrUpdateFirmPolicy;
            PresentInformationToTheUserOrUpdateFirmPolicyEnter();
        }

        //## statechart_method
        public void PresentInformationToTheUserOrUpdateFirmPolicyEnter() {
        }

        //## statechart_method
        public void ForeverInteractWithUserEntDef() {
            animInstance().notifyTransitionStarted("7");
            InteractWithUser_entDef();
            animInstance().notifyTransitionEnded("7");
        }

        //## statechart_method
        public void GetCompetitionInformation_entDef() {
            GetCompetitionInformation_enter();
        }

        //## statechart_method
        public void GetWeatherInformationExit() {
        }

        //## statechart_method
        public void GetWeatherInformation_entDef() {
```

```
            GetWeatherInformation_enter();
        }

        //## statechart_method
        public int ForeverGetMarketInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = ForeverGetMarketInformationTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = OptionalForeverGetMarketInformation_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void OptionalForeverGetMarketInformation_exit() {
            switch (OptionalForeverGetMarketInformation_subState) {
                case ForeverGetMarketInformation:
                {
                    ForeverGetMarketInformation_exit();
                    break;
                }
                case state_23:
                {
                    state_23_exit();
                    break;
                }
                default:
                    break;
            }
            OptionalForeverGetMarketInformation_subState = RiJNonState;
            OptionalForeverGetMarketInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation");
        }

        //## statechart_method
        public void ProductPricingAgent_enter() {
            animInstance().notifyStateEntered("ROOT.ProductPricingAgent");
            rootState_subState = ProductPricingAgent;
            rootState_active = ProductPricingAgent;
            ProductPricingAgentEnter();
        }

        //## statechart_method
        public void state_15Exit() {
        }

        //## statechart_method
        public int DecideOnPricingPolicyTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            //## transition 1
            if(isCompleted(DecideOnPricingPolicy))
                {
                    animInstance().notifyTransitionStarted("1");
                    DecideOnPricingPolicy_exit();
                    DecideOnPricingPolicy_entDef();
                    animInstance().notifyTransitionEnded("1");
                    res = RiJStateReactive.TAKE_EVENT_COMPLETE;
                }
            return res;
        }

        //## statechart_method
        public void InteractWithUserEnter() {
        }

        //## statechart_method
        public void InteractWithUserEntDef() {
            animInstance().notifyTransitionStarted("9");
            PresentInformationToTheUserOrUpdateFirmPolicy_entDef();
```

```
            animInstance().notifyTransitionEnded("9");
        }

        //## statechart_method
        public void GetCompetitionInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation.GetMarketInformation.GetCompetitionInformation")
;
            pushNullConfig();
            GetMarketInformation_subState = GetCompetitionInformation;
            OptionalForeverGetMarketInformation_active = GetCompetitionInformation;
            GetCompetitionInformationEnter();
        }

        //## statechart_method
        public void GetWeatherInformation_exit() {
            popNullConfig();
            GetWeatherInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation.GetMarketInformation.GetWeatherInformation");
        }

        //## statechart_method
        public void state_28_entDef() {
            state_28_enter();
        }

        //## statechart_method
        public void ForeverGetMarketInformation_entDef() {
            ForeverGetMarketInformation_enter();

            animInstance().notifyTransitionStarted("21");
            GetMarketInformation_entDef();
            animInstance().notifyTransitionEnded("21");
        }

        //## statechart_method
        public void state_23_exit() {
            state_23Exit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.state_23");
        }

        //## statechart_method
        public void OptionalForeverGetMarketInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation");
            OptionalForeverGetMarketInformationEnter();
        }

        //## statechart_method
        public void rootState_exit() {
            if(rootState_subState == ProductPricingAgent)
                {
                    ProductPricingAgent_exit();
                }
            rootState_subState = RiJNonState;
            rootStateExit();
            animInstance().notifyStateExited("ROOT");
        }

        //## statechart_method
        public void rootStateEntDef() {
            ProductPricingAgent_entDef();
        }

        //## statechart_method
        public int GetProductsInformationTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("4");
            GetProductsInformation_exit();
            DeterminePricingPolicy_entDef();
```

```
            animInstance().notifyTransitionEnded("4");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void UpdateFirmPolicy_exit() {
            popNullConfig();
            UpdateFirmPolicyExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser.Int
eractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy.UpdateFirmPolicy");
        }

        //## statechart_method
        public void GetCompetitionInformationExit() {
        }

        //## statechart_method
        public void GetLocalInformationEnter() {
        }

        //## statechart_method
        public void GetWeatherInformation_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation.GetMarketInformation.GetWeatherInformation");
            pushNullConfig();
            GetMarketInformation_subState = GetWeatherInformation;
            OptionalForeverGetMarketInformation_active = GetWeatherInformation;
            GetWeatherInformationEnter();
        }

        //## statechart_method
        public void state_28_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.OptionalForeverGetMarketIn
formation.ForeverGetMarketInformation.GetMarketInformation.state_28");
            GetMarketInformation_subState = state_28;
            OptionalForeverGetMarketInformation_active = state_28;
            state_28Enter();
        }

        //## statechart_method
        public void ProductPricingAgentExit() {
        }

        //## statechart_method
        public int GetProductsInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = GetProductsInformationTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = DecideOnPricingPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void state_15_entDef() {
            state_15_enter();
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicyEntDef() {
            animInstance().notifyTransitionStarted("0");
            DecideOnPricingPolicy_entDef();
            animInstance().notifyTransitionEnded("0");
        }

        //## statechart_method
        public int PresentInformationToTheUserTakeNull() {
```

```
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("15");
            PresentInformationToTheUser_exit();
            state_21_entDef();
            animInstance().notifyTransitionEnded("15");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void PresentInformationToTheUser_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverInteractWithUser.In
teractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy.PresentInformationToTheUs
er");
            pushNullConfig();
            PresentInformationToTheUserOrUpdateFirmPolicy_subState =
PresentInformationToTheUser;
            ForeverInteractWithUser_active = PresentInformationToTheUser;
            PresentInformationToTheUserEnter();
        }

        //## statechart_method
        public int state_21_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            res = PresentInformationToTheUserOrUpdateFirmPolicy_takeEvent(id);
            return res;
        }

        //## statechart_method
        public void state_21_exit() {
            state_21Exit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser.Int
eractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy.state_21");
        }

        //## statechart_method
        public void UpdateFirmPolicy_entDef() {
            UpdateFirmPolicy_enter();
        }

        //## statechart_method
        public void PresentInformationToTheUserOrUpdateFirmPolicy_exit() {
            popNullConfig();
            switch (PresentInformationToTheUserOrUpdateFirmPolicy_subState) {
                case PresentInformationToTheUser:
                {
                    PresentInformationToTheUser_exit();
                    break;
                }
                case UpdateFirmPolicy:
                {
                    UpdateFirmPolicy_exit();
                    break;
                }
                case state_21:
                {
                    state_21_exit();
                    break;
                }
                default:
                    break;
            }
            PresentInformationToTheUserOrUpdateFirmPolicy_subState = RiJNonState;
            PresentInformationToTheUserOrUpdateFirmPolicyExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser.Int
eractWithUser.PresentInformationToTheUserOrUpdateFirmPolicy");
        }

        //## statechart_method
        public void PresentInformationToTheUserOrUpdateFirmPolicy_entDef() {
            PresentInformationToTheUserOrUpdateFirmPolicy_enter();

            if(TRUE)
```

```
            {
                animInstance().notifyTransitionStarted("12");
                animInstance().notifyTransitionStarted("14");
                UpdateFirmPolicy_entDef();
                animInstance().notifyTransitionEnded("14");
                animInstance().notifyTransitionEnded("12");
            }
        }

        //## statechart_method
        public void state_28Exit() {
        }

        //## statechart_method
        public void OptionalForeverGetMarketInformation_entDef() {
            OptionalForeverGetMarketInformation_enter();
            OptionalForeverGetMarketInformationEntDef();
        }

        //## statechart_method
        public void DeterminePricingPolicy_entDef() {
            DeterminePricingPolicy_enter();
        }

        //## statechart_method
        public int WaitForNewPeriodTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("3");
            WaitForNewPeriod_exit();
            GetProductsInformation_entDef();
            animInstance().notifyTransitionEnded("3");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public int DecideOnPricingPolicy_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = DecideOnPricingPolicyTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = ForeverDecideOnPricingPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicyExit() {
        }

        //## statechart_method
        public int UpdateFirmPolicyTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("16");
            UpdateFirmPolicy_exit();
            state_21_entDef();
            animInstance().notifyTransitionEnded("16");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void PresentInformationToTheUserOrUpdateFirmPolicyExit() {
        }

        //## statechart_method
        public void state_18Exit() {
        }

        //## statechart_method
        public void state_18_entDef() {
            state_18_enter();
```

```
        }

        //## statechart_method
        public void InteractWithUser_exit() {
            popNullConfig();
            switch (InteractWithUser_subState) {
                case PresentInformationToTheUserOrUpdateFirmPolicy:
                {
                    PresentInformationToTheUserOrUpdateFirmPolicy_exit();
                    break;
                }
                case state_18:
                {
                    state_18_exit();
                    break;
                }
                default:
                    break;
            }
            InteractWithUser_subState = RiJNonState;
            InteractWithUserExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.ForeverInteractWithUser.Int
eractWithUser");
        }

        //## statechart_method
        public void InteractWithUser_entDef() {
            InteractWithUser_enter();
            InteractWithUserEntDef();
        }

        //## statechart_method
        public int ForeverInteractWithUser_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            return res;
        }

        //## statechart_method
        public void ForeverInteractWithUserEnter() {
        }

        //## statechart_method
        public void ForeverInteractWithUser_entDef() {
            ForeverInteractWithUser_enter();
            ForeverInteractWithUserEntDef();
        }

        //## statechart_method
        public void UpdateFactsExit() {
        }

        //## statechart_method
        public void rootStateExit() {
        }

        //## statechart_method
        public int DeterminePricingPolicy_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = DeterminePricingPolicyTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = DecideOnPricingPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void FixPrices_enter() {

animInstance().notifyStateEntered("ROOT.ProductPricingAgent.ForeverDecideOnPricingPoli
cy.DecideOnPricingPolicy.FixPrices");
```

```
            pushNullConfig();
            DecideOnPricingPolicy_subState = FixPrices;
            ForeverDecideOnPricingPolicy_active = FixPrices;
            FixPricesEnter();
        }

        //## statechart_method
        public int InteractWithUser_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = InteractWithUserTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = ForeverInteractWithUser_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public void InteractWithUserExit() {
        }

        //## statechart_method
        public int GetCompetitionInformationTakeNull() {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            animInstance().notifyTransitionStarted("25");
            GetCompetitionInformation_exit();
            UpdateFacts_entDef();
            animInstance().notifyTransitionEnded("25");
            res = RiJStateReactive.TAKE_EVENT_COMPLETE;
            return res;
        }

        //## statechart_method
        public void GetLocalInformation_exit() {
            popNullConfig();
            GetLocalInformationExit();

animInstance().notifyStateExited("ROOT.ProductPricingAgent.OptionalForeverGetMarketInf
ormation.ForeverGetMarketInformation.GetMarketInformation.GetLocalInformation");
        }

        //## statechart_method
        public void ForeverGetMarketInformationEnter() {
        }

        //## statechart_method
        public void state_23_entDef() {
            state_23_enter();
        }

        //## statechart_method
        public void OptionalForeverGetMarketInformationEntDef() {
            if(TRUE)
                {
                    animInstance().notifyTransitionStarted("17");
                    animInstance().notifyTransitionStarted("19");
                    state_23_entDef();
                    animInstance().notifyTransitionEnded("19");
                    animInstance().notifyTransitionEnded("17");
                }
        }

        //## statechart_method
        public void GetProductsInformationExit() {
        }

        //## statechart_method
        public void ForeverDecideOnPricingPolicyEnter() {
        }

        //## statechart_method
        public int PresentInformationToTheUser_takeEvent(short id) {
```

```
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            if(event.isTypeOf(RiJEvent.NULL_EVENT_ID))
                {
                    res = PresentInformationToTheUserTakeNull();
                }

            if(res == RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
                {
                    res = PresentInformationToTheUserOrUpdateFirmPolicy_takeEvent(id);
                }
            return res;
        }

        //## statechart_method
        public int GetMarketInformation_takeEvent(short id) {
            int res = RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
            res = ForeverGetMarketInformation_takeEvent(id);
            return res;
        }

        /**  methods added just for design level debugging instrumentation */
        public boolean startBehavior() {
            try {
              animInstance().notifyBehavioralMethodEntered("startBehavior",
                  new ArgData[] {
                   });
              return super.startBehavior();
            }
            finally {
              animInstance().notifyMethodExit();
            }
        }
        public int takeEvent(RiJEvent event) {
            try {
              //animInstance().notifyTakeEvent(new AnimEvent(event));
              animInstance().notifyBehavioralMethodEntered("takeEvent",
                  new ArgData[] { new ArgData(RiJEvent.class, "event",
event.toString())
                   });
              return super.takeEvent(event);
            }
            finally {
              animInstance().notifyMethodExit();
            }
        }
        /**  see com.telelogic.rhapsody.animation.AnimatedReactive interface */
        public AnimInstance animInstance() {
            return Product_Pricing_Agent.this.animInstance();
        }

    }
    //#[ ignore
    /**  see com.telelogic.rhapsody.animation.Animated interface */
    public AnimClass getAnimClass() {
        return animClassProduct_Pricing_Agent;
    }
    /**  see com.telelogic.rhapsody.animation.Animated interface */
    public Object getFieldValue(java.lang.reflect.Field f, Object userInstance) {
         Object obj = null;
         try {
             obj = f.get(userInstance);
         } catch(Exception e) {
              System.err.println("Exception: getting Field value: " + e);
              e.printStackTrace();
         }
         return obj;
    }
    /**  see com.telelogic.rhapsody.animation.Animated interface */
    public AnimInstance animInstance() {
        if (animate == null)
            animate = new Animate();
        return animate;
    }
    /**  see com.telelogic.rhapsody.animation.Animated interface */
    public void addAttributes(AnimAttributes msg) {
```

```
        msg.add("userClosedGUI", userClosedGUI);
        msg.add("firmStrategy", firmStrategy);
        msg.add("products", products);
        msg.add("productTypes", productTypes);
        msg.add("pricingInterval", pricingInterval);
    }
    /**  see com.telelogic.rhapsody.animation.Animated interface */
    public void addRelations(AnimRelations msg) {

    }
    /** An inner class added as instrumentation for animation */
    public class Animate extends AnimInstance {
        public  Animate() {
            super(Product_Pricing_Agent.this);
        }
        public void addAttributes(AnimAttributes msg) {
            Product_Pricing_Agent.this.addAttributes(msg);
        }
        public void addRelations(AnimRelations msg) {
            Product_Pricing_Agent.this.addRelations(msg);
        }

        public void addStates(AnimStates msg) {
            if ((reactive != null) && (reactive.isTerminated() == false))
              reactive.rootState_add(msg);
        }

    }
    //#]

}
/*********************************************************************
  File Path  : DefaultComponent/DefaultConfig/eu/singularlogic/MARKET-
MINER/Product_Pricing_Agent.java
*********************************************************************/
```

Annex 8.

The Micro Saint Configuration for ASK-IT Project Simulation

Table 7. Micro Saint Entity Attributes

Name	Type	InitialValue	Notes	IsArray
Arrive	double	0.0		FALSE
hasBeenProcessed	boolean	FALSE		FALSE
Performative	string	0		FALSE
Sender	string	0		FALSE
SimpleService	boolean	FALSE		FALSE

Table 8. Micro Saint Variables

Name	Type	IsArray	ArrayDimensions
CPMapRequestMessagesTags	int[]	TRUE	1000
CPPOIsRequestMessagesTags	int[]	TRUE	1000
CPRouteRequestMessagesTags	int[]	TRUE	1000
CPUs	int	FALSE	
TimeSystem	double	FALSE	

Table 9. Micro Saint Scenario Events

Name	Code	StartTime
ScenarioEvent1	CPUs = 2; Entity.Tag=1;	0
ScenarioEvent2	Model.Halt();	1800000

Micro Saint Task Network

Table 10. Micro Saint Release Conditions and Effects

Name	ID	Release Condition	BeginEffect	EndEffect
Personal Assistant	47			
PA Receive Response	47_1	return true;		TimeSystem = Clock-Entity.Arrive;

PA Send Request	47_2	return true;		Entity.Sender="PA";
				Entity.Performative="Request";
				double x;
				x=Model.Random();
				if (x >= .9){
				Entity.SimpleService=true;
				}
				else{
				Entity.SimpleService=false;
				}
				Entity.Arrive = Clock;

Broker Agent	50			
BR Receive Request	50_1	return CPUs>=1;	CPUs-=1;	CPUs+=1;
BR Send Message	50_11	return CPUs>=1;	CPUs-=1;	CPUs+=1;
BR Receive Message	50_12	return CPUs>=1;	CPUs--;	CPUs++;
BR Match	50_2	return CPUs>=1;	CPUs-=1;	CPUs+=1;

BR Invoke DM Service	50_3	return CPUs>=1;	CPUs-=1;	CPUs+=1;
				Entity.Performative="Inform";
				if (Entity.Sender.CompareTo("PA")==0){
				Entity.hasBeenProcessed=true;
				}

BR Send Request	50_4	return CPUs>=1;	CPUs-=1;	CPUs+=1; Entity.Performative="Request"; Entity.Sender = "BR";
BR Receive Response	50_5	return CPUs>=1;	CPUs-=1;	CPUs+=1;
BR Send Response	50_6	return CPUs>=1;	CPUs-=1;	CPUs+=1; Entity.Sender = "BR";
Provider Agent	51			
CP Receive Request	51_1	return CPUs>=1;	CPUs-=1;	CPUs+=1;

CP Send Route Request	51_10	return CPUs>=1;	CPUs-=1;	CPUs+=1;
				Entity.Performative="Request";
				Entity.Sender = "CP";
				for (int i = 0 ; i < CPRouteRequestMessagesTags.Length ; i++) {
				if (CPRouteRequestMessagesTags[i]==0){
				CPRouteRequestMessagesTags[i]=Entity.Tag;
				break;
				}
				}

CP Receive Route Response	51_11	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Sort Routes	51_12	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Send Message	51_15	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Receive Message	51_16	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Decide POI Types	51_2	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Decide Route Type	51_3	return CPUs>=1;	CPUs-=1;	CPUs+=1;

Name	ID	Release Condition	BeginEffect	EndEffect

CP Send POI Request	51_4	return CPUs>=1;	CPUs-=1;	CPUs+=1;
				Entity.Performative="Request";
				Entity.Sender = "CP";
				for (int i = 0 ; i < CPPOIsRequestMessagesTags.Length ; i++) {
				if (CPPOIsRequestMessagesTags[i]==0){
				CPPOIsRequestMessagesTags[i]=Entity.Tag;
				break;
				}
				}

CP Receive POI Response	51_5	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Decide POIs	51_6	return CPUs>=1;	CPUs-=1;	CPUs+=1;

CP Send Map Request	51_7	return CPUs>=1;	CPUs-=1;	CPUs+=1;
				Entity.Performative="Request";
				Entity.Sender = "CP";
				for (int i = 0 ; i < CPMapRequestMessagesTags.Length ; i++) {
				if (CPMapRequestMessagesTags[i]==0){
				CPMapRequestMessagesTags[i]=Entity.Tag;
				break;
				}
				}

CP Receive Map Response	51_8	return CPUs>=1;	CPUs-=1;	CPUs+=1;
CP Send Response	51_9	return CPUs>=1;	CPUs-=1;	CPUs+=1; Entity.Performative="Inform"; Entity.Sender = "CP"; Entity.hasBeenProcessed=true;
start	52	return true;	Entity.Arrive = Clock;	Entity.Tag++;

Table 11. Micro Saint Tasks Timing

Name	ID	Distribution	DS1	DS2	DS3
Personal Assistant	47				
PA Receive Response	47_1	Normal	return 24.0;	return 63.0;	return 0.0;
PA Send Request	47_2	Normal	return 0.0;	return 0.0;	return 0.0;
Broker Agent	50				
BR Receive Request	50_1	Normal	return 24.0;	return 63.0;	return 0.0;
BR Send Message	50_11	Normal	return 2.0;	return 2.0;	return 0.0;
BR Receive Message	50_12	Gamma	return 2.0;	return 2.0;	return 0.0;
BR Match	50_2	Normal	return 254.0;	return 112.0;	return 0.0;
BR Invoke DM Service	50_3	Normal	return 2639.0;	return 1113.0;	return 0.0;
BR Send Request	50_4	Normal	return 7.0;	return 6.0;	return 0.0;
BR Receive Response	50_5	Normal	return 24.0;	return 63.0;	return 0.0;
BR Send Response	50_6	Normal	return 7.0;	return 6.0;	return 0.0;
Provider Agent	51				
CP Receive Request	51_1	Normal	return 24.0;	return 63.0;	return 0.0;
CP Send Route Request	51_10	Normal	return 7.0;	return 6.0;	return 0.0;
CP Receive Route Response	51_11	Normal	return 24.0;	return 63.0;	return 0.0;
CP Sort Routes	51_12	Normal	return 127.0;	return 56.0;	return 0.0;
CP Send Message	51_15	Normal	return 0.0;	return 0.0;	return 0.0;
CP Receive Message	51_16	Normal	return 0.0;	return 0.0;	return 0.0;
CP Decide POI Types	51_2	Normal	return 127.0;	return 56.0;	return 0.0;
CP Decide Route Type	51_3	Normal	return 127.0;	return 56.0;	return 0.0;
CP Send POI Request	51_4	Normal	return 7.0;	return 6.0;	return 0.0;
CP Receive POI Response	51_5	Normal	return 24.0;	return 63.0;	return 0.0;
CP Decide POIs	51_6	Normal	return 127.0;	return 56.0;	return 0.0;
CP Send Map Request	51_7	Normal	return 7.0;	return 6.0;	return 0.0;
CP Receive Map Response	51_8	Normal	return 24.0;	return 63.0;	return 0.0;
CP Send Response	51_9	Normal	return 7.0;	return 6.0;	return 0.0;
start	52	Exponential	return 30000.0;	return 0.0;	return 0.0;

Table 12. Micro Saint Path Decision

Name	ID	Paths	Decision Type	Path 1	Path 2
Personal Assistant	47				
PA Receive Response	47_1				
PA Send Request	47_2				
Broker Agent	50				
BR Receive Request	50_1	50_2			

BR Send Message	50_11		Tactical	if (Entity.hasBeenProcessed == true){
				return true;
				}
				else {
				return false;
				}
					if (Entity.hasBeenProcessed == false) {
					return true;
					}
					else {
					return false;
					}

BR Receive Message	50_12	50_1, 50_5	Tactical	if (((Entity.Sender.CompareTo("CP")==0)
				&&(Entity.Performative.CompareTo(
				"Request") == 0))
				||(Entity.Sender.CompareTo("PA")==0)) {
				return true;
				}
				else {
				return false;
				}
					if ((Entity.Sender.CompareTo("CP")==0)
					&&(Entity.Performative.CompareTo(
					"Request") != 0)) {
					return true;
					}
					else {
					return false;
					}

BR Match	50_2	50_3, 50_4	Tactical	if (Entity.Sender.CompareTo("CP")==0){
				return true;
				}
				else{
				if (Entity.SimpleService==true){
				return true;
				}
				else{
				return false;
				}
				}
					if (Entity.Sender.CompareTo("CP")==0){
					return false;
					}
					else{
					if (Entity.SimpleService==false){
					return true;
					}
					else{
					return false;
					}
					}

BR Invoke DM Service	50_3	50_6			
BR Send Request	50_4	50_11			
BR Receive Response	50_5	50_6			
BR Send Response	50_6	50_11			
Provider Agent	51				
CP Receive Request	51_1	51_2, 51_3	Probabilistic	return .5;	return .5;
CP Send Route Request	51_10	51_15			
CP Receive Route Response	51_11	51_12			
CP Sort Routes	51_12	51_9			
CP Send Message	51_15				

Name	ID	Paths	Decision Type	Path 1	Path 2
CP Receive Message	51_16	51_1, 51_5, 51_11, 51_8	Tactical	if (Entity.Performative.CompareTo("Request") == 0) { return true; } else { return false; }	if (Entity.Performative.CompareTo("Request") != 0){ bool found = false; for (int i = 0 ; i < CPPOIsRequestMessagesTags.Length ; i++) { if (CPPOIsRequestMessagesTags[i]==0) { break; } if (CPPOIsRequestMessagesTags[i] == Entity.Tag){ found = true; } if (found==true){ CPPOIsRequestMessagesTags[i] = CPPOIsRequestMessagesTags[i+1]; } } if (found==true) { return true; } else { return false; } } else { return false; }
CP Decide POI Types	51_2	51_4			
CP Decide Route Type	51_3	51_10			
CP Send POI Request	51_4	51_15			
CP Receive POI Response	51_5	51_6			
CP Decide POIs	51_6	51_7			
CP Send Map Request	51_7	51_15			
CP Receive Map Response	51_8	51_9			
CP Send Response	51_9	51_15			
start	52	52			

Continuing for the task “CP Receive Message” (51_16) that has two more paths:

Name	ID	Paths	Decision Type	Path 3	Path 4
CP Receive Message	51_16	51_1, 51_5, 51_11, 51_8	Tactical	if (Entity.Performative.CompareTo("Request") != 0){ bool found = false; for (int i = 0 ; i < CPRouteRequestMessagesTags.Length ; i++) { if (CPRouteRequestMessagesTags[i]==0) { break; } if (CPRouteRequestMessagesTags[i] == Entity.Tag){ found = true; } if (found==true){ CPRouteRequestMessagesTags[i] = CPRouteRequestMessagesTags[i+1]; } } if (found==true) { return true; } else { return false; } } else { return false; }	if (Entity.Performative.CompareTo("Request") != 0){ bool found = false; for (int i = 0 ; i < CPMapRequestMessagesTags.Length ; i++) { if (CPMapRequestMessagesTags[i]==0) { break; } if (CPMapRequestMessagesTags[i] == Entity.Tag){ found = true; } if (found==true){ CPMapRequestMessagesTags[i] = CPMapRequestMessagesTags[i+1]; } } if (found==true) { return true; } else { return false; } } else { return false; }