Using ASEME Methodology for Model-Driven Agent Systems Development

Nikolaos Spanoudakis[1] and Pavlos Moraitis[2]

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

Abstract. This paper shows how an AOSE methodology, the Agent Systems Engineering Methodology (ASEME), uses state of the art technologies from the Model-Driven Engineering (MDE) domain. We present the Agent Modeling Language (AMOLA) metamodels and the model transformation tools that we developed and discuss our choices. Moreover, the way that non-functional requirements are used throughout the software development lifecycle is discussed and presented with two real-world case studies. Finally, we compare ASEME with a set of existing AOSE methodologies.

1 Introduction

During the last years, there has been a growth of interest in the potential of agent technology in the context of software engineering. A new trend in the Agent Oriented Software Engineering (AOSE) field is that of converging towards the Model-Driven Engineering (MDE) paradigm. Thus, a lot of well known AOSE methodologies propose methods and tools for automating models transformations, such as Tropos [23] and INGENIAS [7], but this is done only for some of the software development phases.

This paper aims to show for the first time how the principles of MDE can be used throughout all the software development phases and how the AOSE community can use three different types of transformations in order to produce new models based on previous models. This approach has been used by the Agent Systems Engineering Methodology (ASEME)[1] [26], [28] and shows how an agent-based system can be incrementally modeled by gradually adding more information at each development phase using the appropriate type of model.

ASEME offers some unique characteristics regarding the used MDE approach. It covers all the classic software development phases (from requirements to implementation) and the transition of one phase to another is done through

---

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

model transformations. It employs three transformation types, i.e. model to model (M2M), text to model (T2M) and model to text (M2T). Thus, the analysts/engineers and developers just enrich the models of each phase with information, gradually leading to implementation. Moreover, the design phase model of ASEME is a statechart [10], a modeling paradigm well known to engineers, which can be implemented using a variety of programming languages or an agent-oriented platform.

Another important aspect of ASEME is the support of documentation of non-functional requirements even from the requirements analysis phase. These are propagated in the analysis phase where they are used for taking managerial decisions and selecting the technologies that will be used for design and development.

This paper presents the ASEME process showing the models transformations between the different development phases. The models that are used by ASEME are defined by the Agent Modeling Language (AMOLA, a first version is presented in [28]). Moreover, it emphasizes on the handling of non-functional requirements by ASEME. The next paragraph provides a background on metamodeling and models transformation followed by the definition of the AMOLA metamodels in section two. The ASEME MDE process is presented in section three discussing the used transformation tools. Section four presents how ASEME tackles the issue of non-functional requirements. In section five we evaluate ASEME using empirical results through the development of two real world systems. Related work is discussed in section six and the paper is concluded in section seven.

1.1 Metamodeling and Models Transformation

Model Driven Engineering (MDE) relies heavily on model transformation [25]. 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. The MDE approach has been argued to contribute to non-functional requirements capture, such as portability, interoperability and reusability [15].

In the software engineering domain a *model* is an abstraction of a software system (or part of it) and a *metamodel* is another abstraction, defining the properties of the model itself. However, even a metamodel is itself a model. In the context of model engineering there is yet another level of abstraction, the *metametamodel*, which is defined as a model that conforms to itself [13].

There are four types of model transformation techniques [16]:

– **Model to Model (M2M)** transformation. This kind of transformation is used for transforming a type of graphical model to another type of graphical model. A M2M transformation is based on the source and target metamodels and defines the transformations of elements of the source model to elements of the target model.

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

In the heart of the model transformation procedure is the Eclipse Modeling Framework (EMF, [3]). Ecore is EMF's model of a model (metamodel). It functions as a metametamodel and it is used for constructing metamodels. It defines that a model is composed of instances of the *EClass* type, which can have attributes (instances of the *EAttribute* type) or reference other EClass instances (through the *EReference* type). Finally, EAttributes can be of various *EDataType* instances (such are integers, strings, real numbers, etc). EMF allows to extend existing models via inheritance, using the *ESuperType* relationship for extending an existing EClass.

A similar technology, the Meta-Object Facility (MOF), is an OMG standard [19] for representing metamodels and manipulating them. MOF is older than EMF and it influenced its design. However, the EMF meta-model is simpler than the MOF meta-model in terms of its concepts, properties and containment structure, thus, the mapping of EMF's concepts into MOF's concepts is relatively straightforward and is mostly 1-to-1 translations [8]. EMF is also used today by a large open source community becoming a de facto standard in MDE.

2   The AMOLA Metamodels

In this section we present the metamodels used in the ASEME MDE process. Using these metamodels we can derive graphical tools for defining the models and tools for automating the models tranformations.

2.1   The System Actor Goal Model (SAG)

The SAG model is a subset of the Actor model of the Tropos ecore model [31]. Tropos is, on one hand, one of the very few AOSE methodologies that deal with requirements analysis, and, on the other hand it borrows successful practices from the general software engineering discipline. This is why we have been inspired by Tropos. The reason for not using the Tropos diagrams as they are is
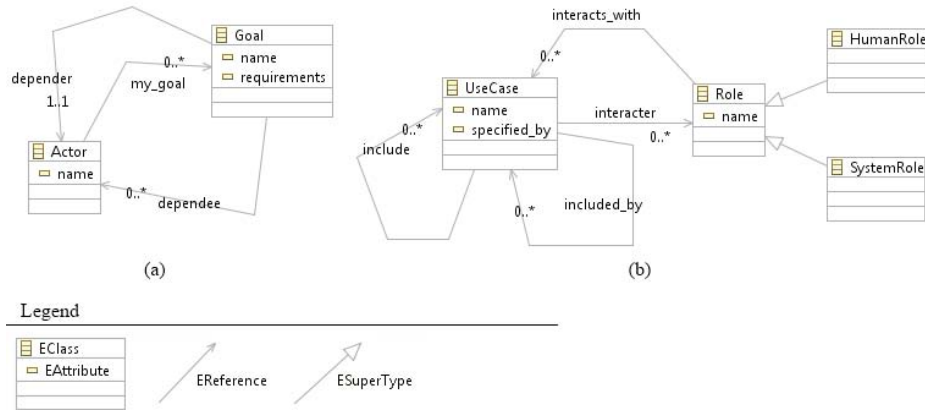
Fig. 1. The AMOLA SAG (a) and SUC (b) metamodels

that they provide more concepts than the ones used by AMOLA as they are also used for system analysis. However, as we will show later, AMOLA defines more well-suited diagrams for system analysis.

Thus, the AMOLA System Actors Goals diagram is the one that appears in Figure 1(a) employing the *Actor* and *Goal* concepts. The actor references his goals using the *EReference my_goal*, while the *Goal* references a unique *depender* and zero or more *dependees*. The reader should notice the choice to add the *requirements EAttribute* of *Goal*. Through this attribute, each goal is related to functional and non-functional requirements, which are documented in plain text form.

2.2 The System Use Cases Model (SUC)

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

In AMOLA no new elements are needed other than those proposed by UML, however, the semantics change. Firstly, the actor "enters" the system and assumes a role. *Agents* are modeled as roles, either within the system box (for the agents that are to be developed) or outside the system box (for existing agents in the environment). Human actors are represented as roles outside the system box (like in traditional UML use case diagrams). This approach aims to show the concept that we are modeling artificial agents interacting with other artificial agents or human agents. Secondly, the different use cases must be directly related to at least one artificial agent role.

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

## 2.3   The System Roles Model (SRM)

An important concept in AOSE is the role. An agent is assumed to undertake one or many roles in his lifetime. The role is associated with activities and this is one of the main differences with traditional software engineering, the fact that the activity is no longer associated with the system, but, rather, with the role. Moreover, after defining the capabilities of the agents and decomposing them to simple activities in the SUC model we need to define the dynamic composition of these activities by each role so that he achieves his goals. Thus, we defined the SRM model based on the Gaia Role model [34]. Gaia defines the liveness formula operators that allow the composition of formulas depicting the role's dynamic behavior. However, we needed to change the role model of Gaia in order to accommodate the integration in an agent's role the incorporation of complex agent interaction protocols (within which an agent can assume more than one roles even at the same time), a weakness of the Gaia methodology. The AMOLA SRM metamodel is presented in Figure 2(a). The SRM metamodel defines the concept *Role* that references the concepts:

- *Activity*, that refers to a simple activity with two attributes, name (its name) and functionality (the description of what this activity does)
- *Capability* that refers to groups of activities (to which it refers) achieving a high level goal, and,
- *Protocol*. The protocol attributes *name* and *participant* refer to the relevant items in the Agent Interactions Protocol (AIP) model. This model is not detailed here-in. It is used for identifying the roles that participate in a protocol, their activities within the protocol and the rules for engaging (for more details consult [29]).

The *Role* concept also has the *name* and *liveness* attributes (the first is the role name and the second its liveness formula). The reader should note the *functionality* attribute of the *Activity* concept which is used to associate the activity to a generic functionality. For example, the "get weather information" activity can be related to the "web service invocation" functionality (see [27], [28]).

2.4   The Intra-Agent Control Model (IAC)

In order to represent system designs, AMOLA is based on statecharts, a well-known and general language and does not make any assumptions on the ontology, communication model, reasoning processes 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 [11]), or strongly imply
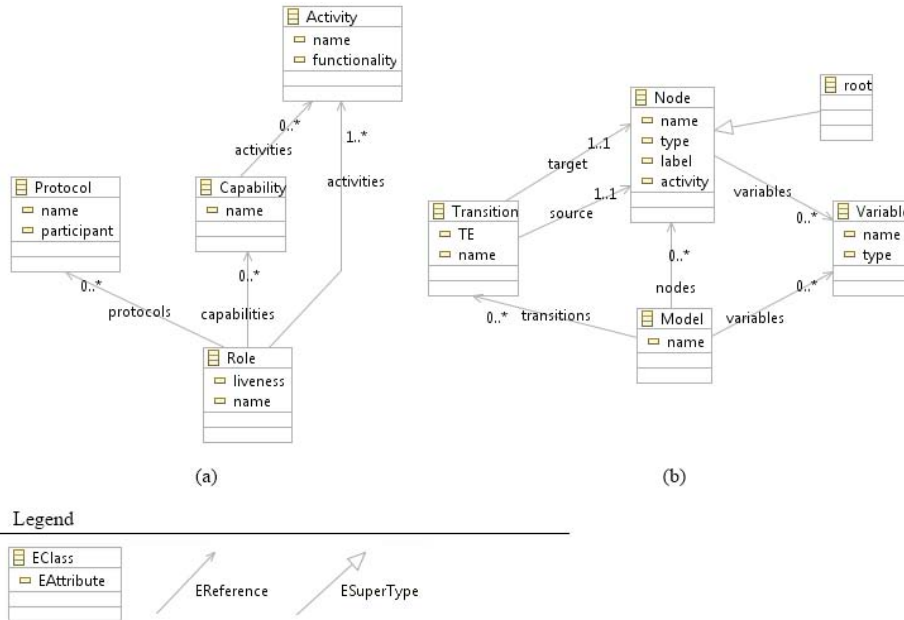
Fig. 2. The AMOLA SRM (a) and IAC (b) metamodels

(like Tropos [11]) the agent mental models. Of course, there are some developers who want to have all these things ready for them, but there are others who want to use different agent paradigms according to their expertise. For example, one can use AMOLA for defining Belief-Desire-Intentions based agents, while another for defining procedural agents [26].

The inspiration for defining the IAC metamodel mainly came from the UML statechart definition. Aiming to define the statechart using the AMOLA definition of statechart [30], the IAC metamodel differs significantly from the UML statechart. However, a UML statechart can be transformed to an IAC statechart, although some elements would be difficult to define (UML does not cater for transition expressions and association of variables to nodes and uses statecharts to define a single object's behaviour). Thus, the IAC metamodel, which is presented in Figure 2(b), defines a *Model* concept that has *nodes*, *transitions* and *variables* EReferences. Note that it also has a *name* EAttribute. The latter is used to define the namespace of the IAC model. The namespace should follow the Java or C# package namespace format. The nodes contain the following attributes:

- *name.* The name of the node,
- *type.* The type of the node, corresponding to the type of state in a statechart, typically one of AND, OR, BASIC, START, END (see [10]),
- *label.* The node's label, and
- *activity.* The activity related to the node.

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

- *name*, usually in the form <source node label> TO <target node label>
- *TE, the transition expression*. This expression contains the conditions and events that make the transition possible. Through the transition expressions (TEs) the modeler defines the control information in the IAC. TEs can use concepts from an ontology as variables. Moreover, the receipt or transmission of an inter-agent message can be used (in the case of agent interaction protocols). For the formal definition of the TE and some examples see [26] or [29].
- *source*, the source node, and,
- *target*, the target node.

3 The ASEME Model-Driven Process and Tools

ASEME is described in detail in [26]. It is a complete process incorporating all the traditional software engineering methodology phases, however, using the SPEM 2.0 process metamodel [21] it can be modified to provide an agile process. Figure 3, a screenshot from the EPF[2] modelling tool, shows on the left side the ASEME method library and its various properties. From top to bottom the most important are the:

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

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

1. *Edit SAG model*. The business consultant of the software development firm identifies the actors involved in the system to be along with their goals.

---

[2] The Eclipse Process Framework (EPF) aims at producing a customizable software process engineering framework. URL: `http://www.eclipse.org/epf/`
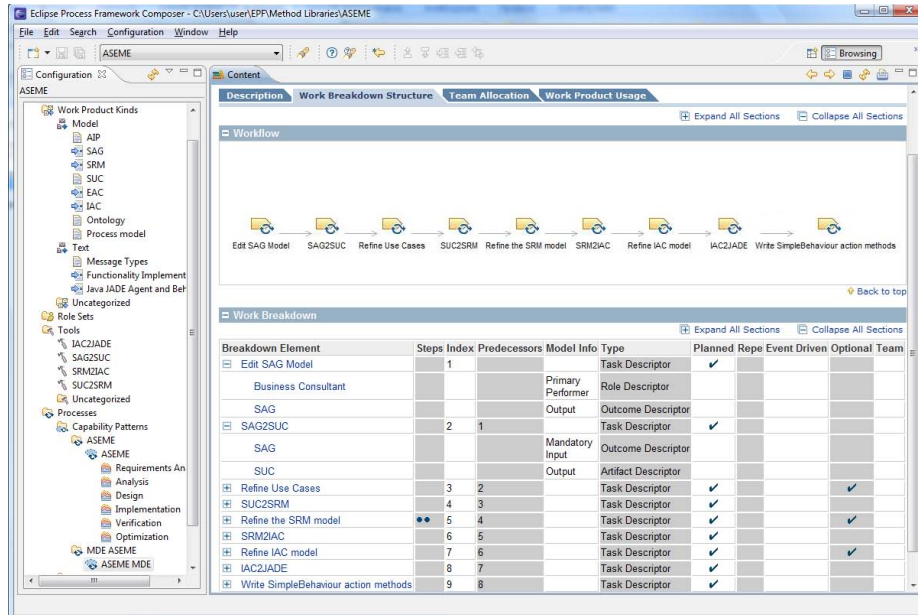
Fig. 3. The ASEME MDE Process

2. *SAG2SUC.* An automated task, as the reader can see in the figure this task has only a mandatory input model (SAG) and an output model (SUC). It creates an initial SUC model based on the previously created SAG model.
3. *Refine Use Cases.* The analyst works on the SUC model and refines the general use cases using the include relationship. He/she also identifies which actors will be implemented defining them as human or artificial agent actors. The overall system design is enriched by identifying the tasks that have to be carried out by the actors.
4. *SUC2SRM.* An automated task, it has only a mandatory input model (SUC) and an output model (SRM). It creates an initial SRM model based on the previously created SUC model.
5. *Refine the SRM model.* The analyst works on the SRM model by defining the liveness formulas that will describe the dynamic compilation of the previously identified tasks.
6. *SRM2IAC.* An automated task, it has only a mandatory input model (SRM) and an output model (IAC). It creates multiple initial IAC models based on the previously created SRM model, one for each role.
7. *Refine the IAC model.* The designer works on each IAC model by defining the conditions and/or events that will enable the transitions from one task to the other.

8. *IAC2JADE.* An automated task, it has only a mandatory input model (IAC) and an output model (Java JADE³ Agent and Behaviours code). It creates a JADE Agent class and multiple JADE Behaviour classes for each IAC model.
9. *Write SimpleBehaviour action methods.* The programmer writes code only for the JADE *SimpleBehaviour* class descendants'*action* methods.

The following paragraphs discuss the employed transformations automation tools that are used in the presented ASEME MDE process.

3.1   The ASEME M2M Transformation Tools (SAG2SUC and SUC2SRM)

The Atlas Transformation Language (ATL) [14] was used for model to model (M2M) transformations. Another alternative to ATL would be the Query-View Transformation (QVT) language [20], however, ATL was better documented on the internet with a user guide and examples, while the only resource located for QVT was a presentation. Therefore, and as the requirements of both languages (ATL and QVT) are the same the decision was to choose the better documented one. Such transformations are the SAG2SUC and SUC2SRM.

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

3.2   The ASEME T2M Transformation Tool (SRM2IAC)

The trick in text to model transformations is to define the meta-model of the text to be transformed. This can be done in the form of an EBNF syntax (for languages with a grammar) or through string manipulation. Efftinge and Völter [6] presented the xtext framework in the context of the Eclipse Modeling Project

---

³ The Java Agent Development Environment (JADE) is an open source framework that adheres to the FIPA standards, URL: `http://jade.tilab.com`
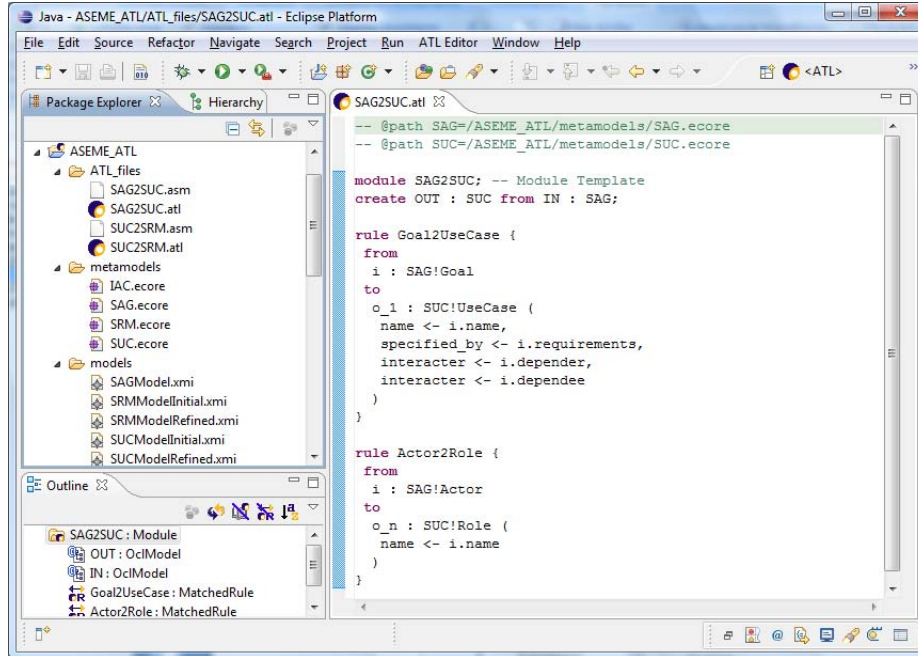
Fig. 4. The eclipse ATL project for the SAG2SUC and the SUC2SRM M2M transformations

(EMP[4]). According to their work, an xText grammar is a collection of rules. Each rule is described using sequences of tokens. Tokens either reference another rule or one of the built-in tokens (e.g. STRING, ID, LINE, INT). A rule results in a meta type, the tokens used in the rule are mapped to properties of that type. xText is used to automatically derive the meta model from the grammar. Then a textual representation of a model following this grammar can be parsed and the meta-model is automatically generated.

Rose et al. [24] described an implementation of the Human-Usable Textual Notation (HUTN) specification of OMG [18] using Epsilon, which is a suite of tools for MDE. OMG created HUTN aiming to offer three main benefits to MDE: a) a generic specification that can provide a concrete HUTN language for any model, b) the HUTN languages to be fully automated both for production and parsing, and, c) the HUTN languages to conform to human-usability criteria. The HUTN implementation automates the transformation process by eliminating the need for a grammar specification by auto defining it accepting as input the relevant EMF meta-model. This is the main reason for choosing HUTN for ASEME.

---

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

A T2M transformation is used for transforming a liveness formula to a state-chart (IAC model). We first use an iterative algorithm (see [30]) that creates the HUTN model, which is then automatically transformed to an IAC model. The usage of the HUTN technology also helped a lot in debugging the algorithm as the output was in human-readable format.

3.3   The ASEME M2T Transformation Tool (IAC2JADE)

The last transformation type used in the ASEME process is M2T. The platform independent IAC model must be transformed to a platform dependent one and to executable code.

We used the Xpand language offered by the Eclipse. Another commonly used M2T transformation language (in EMP) is the Java Emitter Templates (JET). JET uses JSP-like templates, thus it is easy to learn for developers familiar with this technology.

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

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

4   Non-functional Requirements in ASEME

Throughout this section, and aiming to present the way that ASEME handles non-functional requirements, some parts of the requirements and system analysis of two real-world agent-based systems are presented.

4.1   A Real World Case Study: The ASK-IT Project

In this first case study, 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[5] 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

---

[5] The OSGi (Open Services Gateway initiative) Alliance is a worldwide consortium of technology innovators that advances a proven and mature process to assure interoperability of applications and services based on its component integration platform, URL: `http://www.osgi.org`

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 [17].

A subset of the SAG model capturing the ASK-IT system requirements is presented in Figure 5. This model was created after identifying the stakeholders relevant to this project [26]. Such were 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. 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.
- *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.

The stakeholders are modeled as actors. A stakeholder, who is assisted by a software, introduces a new actor, usually named as *personal assistant*. Thus,
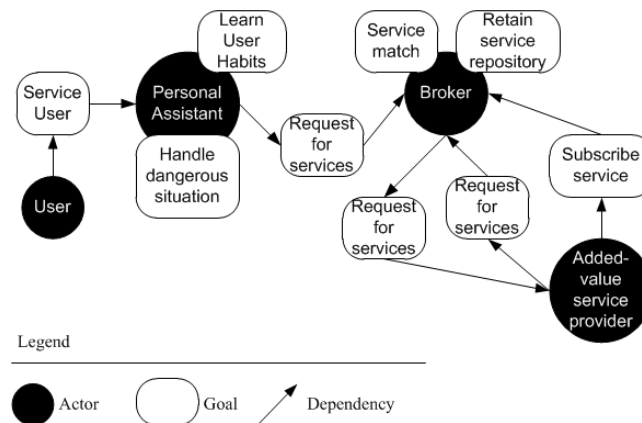
Fig. 5. The SAG model for the ASK-IT project

in Figure 5, 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 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 the "Added-value service provider" (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.

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)

The *requirements per goal* are documented in the *requirements* EAttribute of the *Goal* concept of the SRM model, see Figure 2(a). 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 1).

The SUC model presented in Figure 6 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

Table 1. A portion of the Requirements Per Goal (RPG) model for the Personal Assistant Actor in ASK-IT project

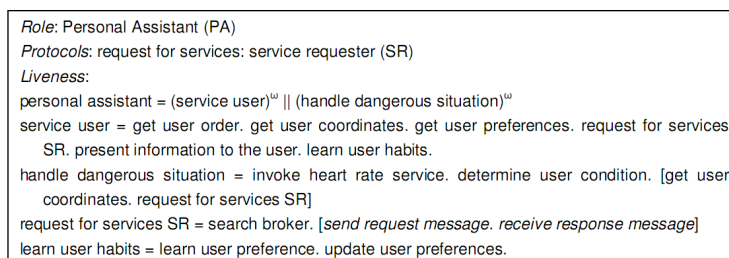| Personal Assistant goals	
Service User	Delivery of the service within 10 seconds
	The service is offered from a mobile device with the OSGi service architecture
	The user can request a mapping or a routing service
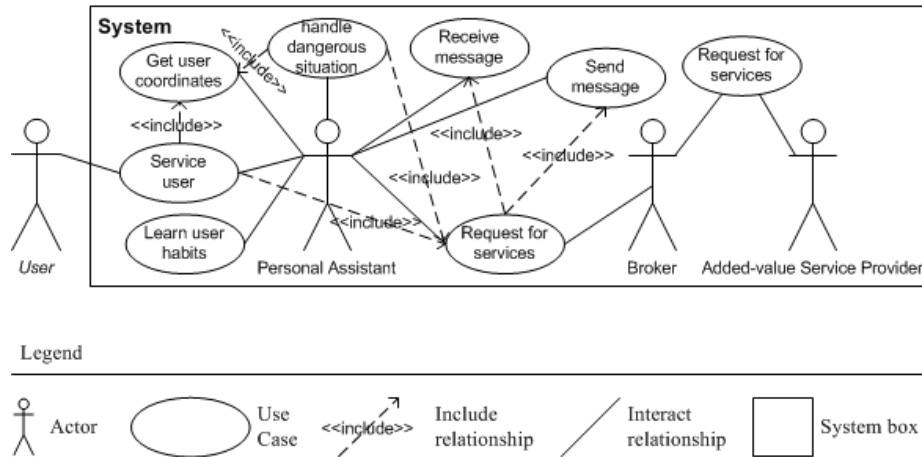
Fig. 6. A portion of the SUC model of the ASK-IT project

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.

A portion of the SRM for the personal assistant (PA) is presented in Figure 7. 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 whose left hand side terms are named after them. Other capabilities are further detailed in following formulas.

Role: Personal Assistant (PA)

Protocols: request for services: service requester (SR)

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.

Fig. 7. A portion of the SRM model of the Personal Assistant role of the ASK-IT project
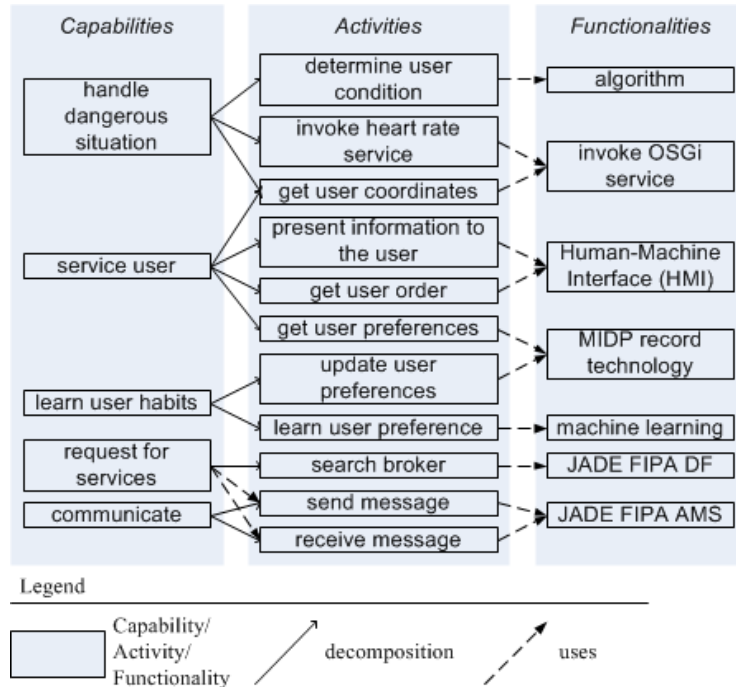
Fig. 8. Functionality Table for the personal assistant role of the ASK-IT project

The Functionality Table (FT) 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 8 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 FIPA[6]. This is the point where the analyst proposes the use of a platform for instantiation, e.g., in our example, JADE. 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.

The functionality table is defined in the SRM model adding a "functionality" property to each activity. However, a decision maker would prefer to see this information in a tabular format (like in Figure 8) in order to gain a quick understanding about the technologies involved in developing each agent.

---

[6] The Foundation for Intelligent Physical Agents (FIPA) is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies, URL: `http://www.fipa.org`

4.2   A Real World Case Study: The Market-Miner Project

In the Market-Miner project [27], we developed an autonomous product pricing agent situated in a firm monitoring for changes of the prices of competitors along with changes in firm policies and deciding on the prices of the products on the self.

During the analysis phase we identified the actors and the use cases related to our agent system. We documented these findings using the ASEME System Use Cases (SUC) model (see Figure 9). For our system, the system actor is the Market-miner Product Pricing Agent (or 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, like in the case of umbrellas) and municipality systems (as local events like concerts, sports, etc, also influence consumer demand).

Having defined the involved actors we started identifying general use cases (like *interact with user*) and then we elaborated them in more specific ones (like *present information to the user* and *update firm policy*) using the ≪ include ≫ relation. After refining the use cases, the SUC model was transformed to the System Roles Model (SRM), see Figure 10(a).
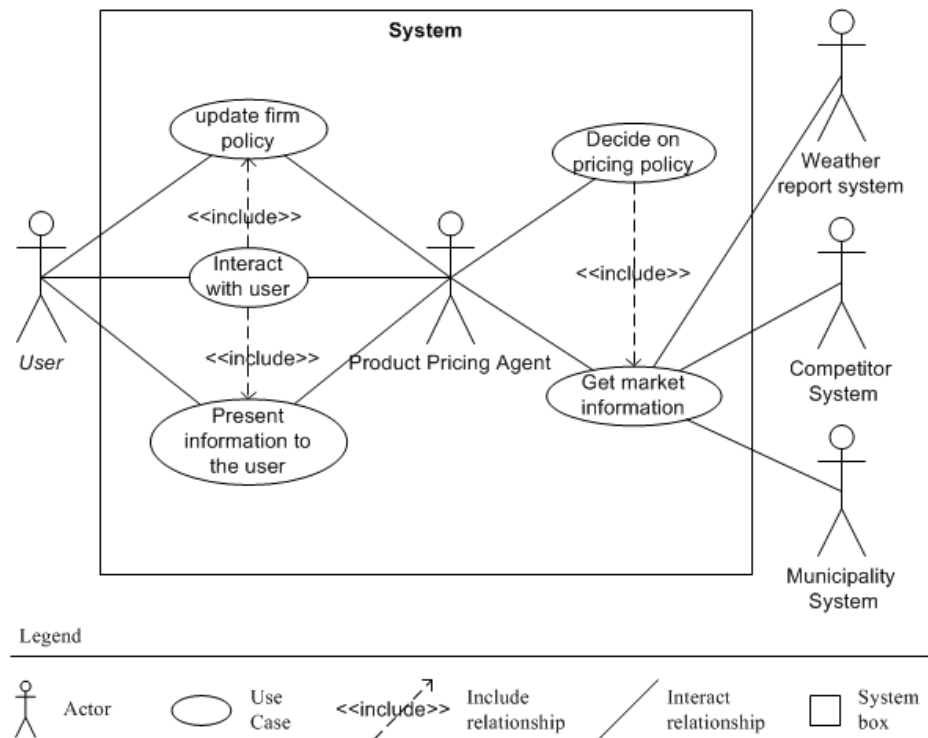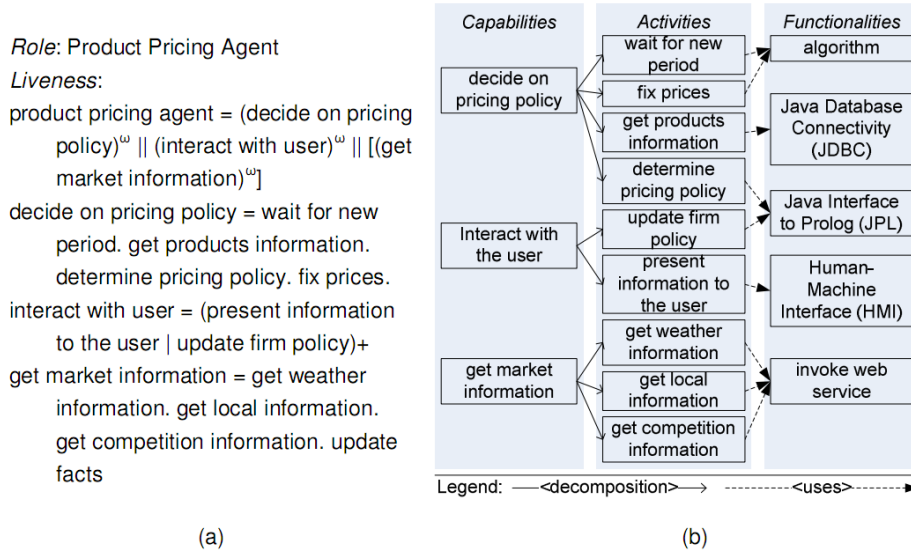

Fig. 9. An extract from the MIPA System SUC Model

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)

Fig. 10. MIPA Role Model (a) and the Functionality Table (b)

The next step was to associate each activity to a functionality, i.e. the technology that would be used for its implementation. In Figure 10(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[7] technology was selected, as it is database provider independent. Moreover, the different information channels that are currently used depend on the same functionality, i.e. a web service invocation. Thus, in the future, new information channels such as a financial channel where from to get relevant news, such as a financial crisis, can be integrated in the system using the same functionality. In this way, this model allows for the easy extensibility of the system (another usually desired non-functional requirement).

5   ASEME Evaluation

For evaluating our work we used two case studies on the development of two real-world systems. The first, a module of the ASK-IT project [17], included programming for semantic service matching and interfaces to other modules that were based on OSGi, a service oriented architecture. The ontology was developed using

---

[7] The Java Database Connectivity (JDBC) is a standard for database-independent connectivity between the Java programming language and a wide range of databases providing a call-level API for SQL-based database access, URL: `http://java.sun.com/javase/technologies/database/`

the Protégé[8] ontology editor and its beangenerator add-on, which generates java files representing an ontology that can be used with the JADE environment. The second is the Market-Miner project [27], where we used Prolog for implementing the decision making capability of the agent. Again, we used the Protégé editor for creating the ontology.

These projects used different implementation platforms, the first the JADE platform, while the second a Java CASE (Computer-Aided Software Engineering) tool, IBM Rational Rhapsody (URL: *http://www.ibm.com*). For the latter it was needed to transform the SRM model to an IAC model manually (as Rhapsody does not offer an import tool for statechart models) using the process defined in [30].

Table 2 shows a quick comparison of ASEME with existing AOSE methodologies. It has been inspired by a similar table in [32] from which we use some criteria (rows). The first row shows the levels of abstraction supported by the methodologies. Only ASEME maintains three levels of abstraction throughout the software development phases. Some do not support abstraction at all, while others do a phase-based abstraction (e.g. define agent interactions and roles in the analysis phase and focus in the specific agent development in the design phase). The next row shows the MDE support for the different software development phases. ASEME supports all the phases, many methodologies support some phases and INGENIAS allows the modeler to define his own transformations. The third row shows if a methodology covers all the software development phases, i.e. requirements analysis, system analysis, design, implementation, verification and optimization. The forth row shows what kind of agents each methodology supports and the fifth row indicates which methodologies define an intra-agent control model that allows an agent to coordinate his capabilities, thus supporting a modular development approach. The sixth row shows that ASEME is the only methodology to use a uniform representation of inter-agent protocols and the intra-agent control allowing for an easy integration of protocols in an agent specification. In Table 2 "n/a" means not applicable.

The last two rows are related to the non-functional requirements capture capability of the methodologies. Only three of the reviewed methodologies address this issue as the seventh row suggests. In Tropos, NFRs are either operationalized (also in MaSE) or are transformed to operational rules in the Late Requirement Analysis phase. Finally, only in ASEME they are used for selecting implementation technologies and tools in the analysis phase while in Tropos they are used for offering alternatives of implementations (also in the form of redundant sub-systems).

6   Related Work

A number of works in AOSE have introduced concepts and ideas from the model-driven engineering domain. Most of them just introduce an MDE technique for

---

[8] Protégé is a free, open source ontology editor and knowledge-base framework, URL: `http://protege.stanford.edu`

Table 2. ASEME compared with existing AOSE methodologies

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

transforming one of their models to another in one phase, e.g. from a Tropos plan decomposition diagram to a UML activity diagram in [23] and from a BDI (Belief-Desire-Intention) representation in XML format to JACK platform code in [12]. Almost all AOSE methodologies define a single, usually huge metamodel covering all the requirements, analysis and design phases [1].

Other works aim to create a single meta-model that can be used by different AOSE methodologies in a specific phase, like in [9], where the authors defined a meta-model (PIM4Agents) that can be used to model MAS in the PIM level of MDA, and in [1], where the authors try to envisage a unifying MAS meta-model. Finally, a more recent work [7] presents an algorithm to generate model transformations by-example that allows the engineer to define himself the transformations that he wants to apply to models complying with the INGENIAS metamodel.

ASEME furthers the state of the art by being the first AOSE methodology to propose a model-driven approach covering all the development phases. Thus, the developer only at the requirements analysis phase starts a model from scratch (SAG). All the other models are launched through a transformation that initializes them. Then, the developer adds the new information related to the specific model.

Regarding the use of non-functional requirements (NFRs), Tropos [2] 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).

Another approach is that proposed by Danny Weyns [33]. In his work, the author addresses the issue of NFRs by selecting appropriate architectures that

each addresses a family of NFRs. For example, he proposes that selecting an agent-based approach to software development contributes to the NFRs openness, adaptability and scalability. Moreover, additional NFRs are modeled in quality attribute scenarios. These consist of three parts, a) *stimulus*: an event occuring in the system, b) *environment*: the environment conditions at the time of the stimulus occurence, and, c) *response*: the activity to be executed when the stimulus arrives.

In ASEME the way to cater for NFRs has been influenced by the work of Pérez et al. [22], who believe that non-functional requirements need a way to influence the way to implement a system or task, and this is what ASEME uniquely achieves compared to the other AOSE methodologies. In ASEME we do not define specific situations as NFRs, we allow quality requirements to be inserted in each goal requirements in the SAG model. Thus, they influence all analysis phase decisions including the technology selection for achieving the goal. Moreover, ASEME could allow for the catering of quality attribute scenarios if they are defined as agent interaction protocols (which define preconditions and results along with the different interested actors activities).

7   Conclusion

In the previous sections, we presented the formal definition of the AMOLA metamodels, which have been inspired by previous works but are original in the way that they uniquely extend those works and insert new semantics, thus assisting the ASEME process. We also presented the models transformations that occur in the different phases of ASEME.

The platform independent model of ASEME, i.e. the IAC, is a statechart which can be transformed to a platform specific model in C++ or Java (using commercial CASE tools) or in the JADE agent platform. This is another originality of ASEME, it is the first AOSE methodology to provide a PIM model that is compatible with existing software tools (i.e. the statechart) giving multiple platform choices to the developers.

The models used are common in the software engineering community, which means that any engineer can quickly adapt to the ASEME process. Model transformations are automated throughout the software development process.

Moreover, ASEME documents quality or non-functional requirements at the requirements analysis phase and allows these to influence the architectural decisions of the analyst(s) when selecting technologies (e.g. reasoning, communication, etc) for realising system tasks. The possiblity of the ASEME IAC model to be transformed to a process model allows for simulating the analysis model even before design and validate the system functional and several non-functional requirements including scalability and robustness.

ASEME has been successfully used for the development of two real world systems ([17], [27]) and is currently used for development of a robotic system [4]. Moreover, we are working on automating the transformation of the IAC model to a process model as there are a number of existing tools in the market that perform system simulation, verification and optimization on such models. In [5]

we proposed transformation templates for doing this transformation manually and performed simulations that showed that the ASK-IT system could deliver the service to the end user in 10 seconds (thus achieving a non-functional requirement, see Table 1).

References

1. Bernon, C., Cossentino, M., Pavón, J.: Agent-oriented software engineering. Knowledge Eng. Review 20(2), 99–116 (2005)
2. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8, 203–236 (2004)
3. Budinsky, F., Brodsky, S.A., Merks, E.: Eclipse Modeling Framework. Pearson Education, London (2003)
4. Chatzilaris, E., Kyranou, I., Orfanoudakis, E., Paraschos, A., Vazaios, E., Spanoudakis, N., Vlassis, N., Lagoudakis, M.G.: Kouretes 2010 spl team description paper. In: RoboCup 2010 Team Description Papers, Singapore (2010)
5. Delias, P., Spanoudakis, N.: Simulating multi-agent system designs using business process modeling. In: Proceedings of the 8th European Workshop on Multi-Agent Systems (EUMAS 2010), Paris, France, December 16-17 (2010)
6. Efftinge, S., Völter, M.: oaw xtext: A framework for textual dsls. In: Eclipse Summit 2006 Workshop: Eclipse Modeling Symposium (2006), `http://www.eclipsecon.org/summiteurope2006/`
7. García-Magariño, I., Rougemaille, S., Fuentes-Fernández, R., Migeon, F., Gleizes, M.P., Gómez-Sanz, J.J.: A tool for generating model transformations by-example in multi-agent systems. In: Demazeau, Y., Pavón, J., Corchado, J.M., Bajo, J. (eds.) 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), Salamanca, Spain, March 25-27. Advances in Soft Computing, vol. 55, pp. 70–79. Springer, Heidelberg (2009)
8. Gerber, A., Raymond, K.: Mof to emf: there and back again. In: Burke, M.G. (ed.) Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange, Anaheim, CA, USA, pp. 60–64. ACM, New York (2003)
9. Hahn, C., Madrigal-Mora, C., Fischer, K.: A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems 18(2), 239–266 (2009)
10. Harel, D., Kugler, H.: The rhapsody semantics of statecharts (or, on the executable core of the uml) - preliminary version. In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) INT 2004. LNCS, vol. 3147, pp. 325–354. Springer, Heidelberg (2004)
11. Henderson-Sellers, B., Giorgini, P.: Agent-oriented methodologies. Idea Group Pub., USA (2005)
12. Jayatilleke, G.B., Padgham, L., Winikoff, M.: A model driven component-based development framework for agents. Comput. Syst. Sci. Eng. 20(4) (2005)
13. Jouault, F., Bézivin, J.: Km3: A dsl for metamodel specification. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 171–185. Springer, Heidelberg (2006)

14. Jouault, F., Kurtev, I.: Transforming models with atl. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
15. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
16. Langlois, B., elena Jitia, C., Jouenne, E.: Dsl classification. In: 7th OOPSLA Workshop on Domain-Specific Modeling (2007)
17. Moraitis, P., Spanoudakis, N.I.: Argumentation-based agent interaction in an ambient-intelligence context. IEEE Intelligent Systems 22(6), 84–93 (2007)
18. OMG: Human-Usable Textual Notation V1.0 (2004)
19. OMG: Meta Object Facility (MOF) Core Specification Version 2.0 (2006), `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`
20. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.0 (2008), `http://www.omg.org/spec/QVT/1.0/PDF/`
21. OMG: Software and Systems Process Engineering Meta-Model Specification, version 2.0 (2008)
22. Pérez, F.J., Laguna, M.A., González-Carvajal, Y.C., González-Baixauli, B.: Requirements variability support through mda^{tm} and graph transformation. Electr. Notes Theor. Comput. Sci. 152, 161–173 (2006)
23. Perini, A., Susi, A.: Automating model transformations in agent-oriented modelling. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 167–178. Springer, Heidelberg (2006)
24. Rose, L.M., Paige, R.F., Kolovos, D.S., Polack, F.: Constructing models with the human-usable textual notation. In: Busch, C., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 249–263. Springer, Heidelberg (2008)
25. Sendall, S., Kozaczynski, W.: Model transformation: The heart and soul of model-driven software development. IEEE Software 20(5), 42–45 (2003)
26. Spanoudakis, N.: The Agent Systems Engineering Methodology (ASEME). Ph.D. thesis, Paris Descartes University (2009)
27. Spanoudakis, N., Moraitis, P.: Engineering an agent-based system for product pricing automation. Engineering Intelligent Systems for Electrical Engineering and Communications 17(2-3), 139–151 (2009)
28. Spanoudakis, N.I., Moraitis, P.: The agent modeling language (amola). In: Dochev, D., Pistore, M., Traverso, P. (eds.) AIMSA 2008. LNCS (LNAI), vol. 5253, pp. 32–44. Springer, Heidelberg (2008)
29. Spanoudakis, N.I., 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 2008), Sydney, NSW, Australia, December 9-12, pp. 578–582. IEEE, Los Alamitos (2008)
30. Spanoudakis, N.I., Moraitis, P.: Gaia agents implementation through models transformation. In: Yang, J.-J., Yokoo, M., Ito, T., Jin, Z., Scerri, P. (eds.) PRIMA 2009. LNCS, vol. 5925, pp. 127–142. Springer, Heidelberg (2009)
31. Susi, A., Perini, A., Mylopoulos, J., Giorgini, P.: The tropos metamodel and its use. Informatica (Slovenia) 29(4), 401–408 (2005)
32. Tran, Q., Low, G.: Comparison of ten agent-oriented methodologies. In: Agent-oriented methodologies [11]
33. Weyns, D.: Architecture-Based Design of Multi-Agent Systems, 1st edn. Springer Publishing Company, Heidelberg (2010) (incorporated)
34. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Trans. Softw. Eng. Methodol. 12(3), 317–370 (2003)