Model-Driven Agents Development with ASEME

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

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

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

Keywords: Model Driven Engineering, Agent Oriented Software Engineering

1 Introduction

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

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

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

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

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

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

2 Metamodeling and Models Transformation

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

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

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

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

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

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

3 The AMOLA Metamodels

System Actor Goal model (SAG)

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

Use case model (SUC)

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

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

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

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

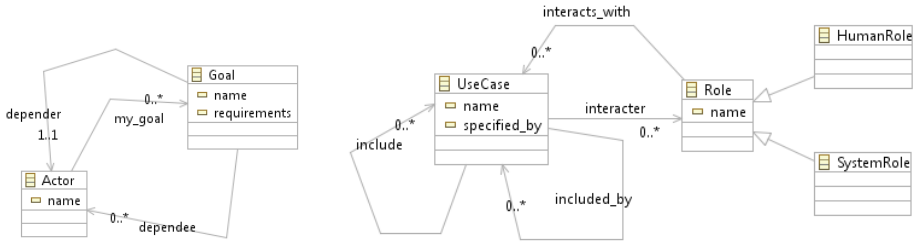

Fig. 1. The AMOLA SAG (a) and SUC (b) metamodels

Role model (SRM)

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

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

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

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

Intra-agent control model (IAC)

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

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

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

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

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

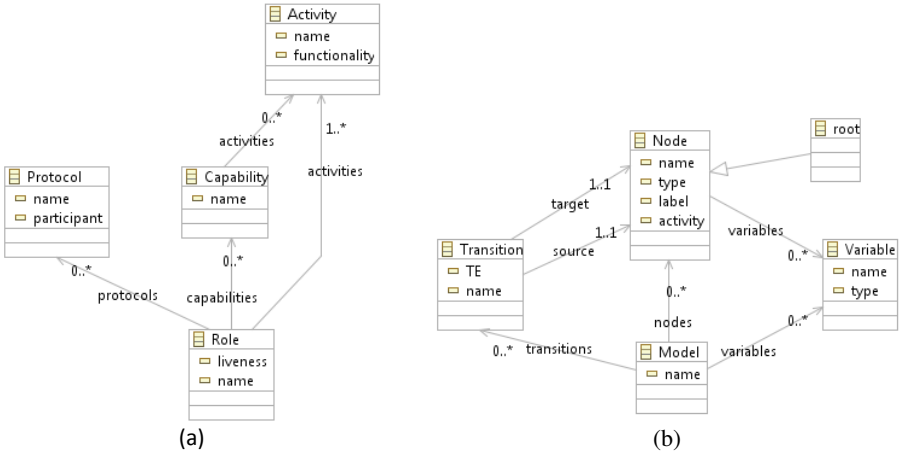

Fig. 2. The AMOLA SRM (a) and IAC (b) metamodels.

4 The ASEME Model-Driven Process and Tools

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

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

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

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

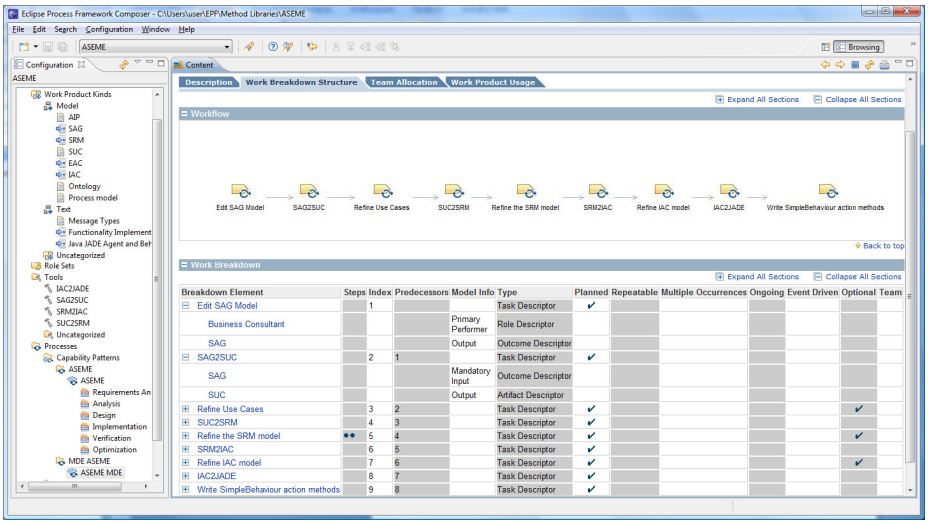

Fig. 3. The ASEME MDE Process

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

ASEME M2M Transformation Tools (SAG2SUC and SUC2SRM)

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

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

ASEME T2M Transformation Tool (SRM2IAC)

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

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

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

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

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

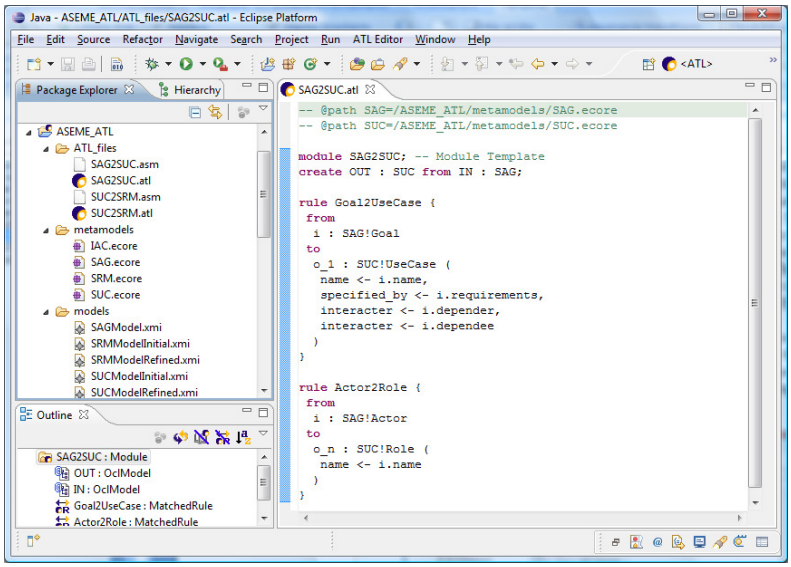

Fig. 4. The eclipse ATL project for the SAG2SUC and the SUC2SRM M2M transformations.

ASEME M2T Transformation Tool (IAC2JADE)

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

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

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

5 Related Work and Conclusion

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

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

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

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

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

Table 1. ASEME compared with existing AOSE methodologies.

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

References

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

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