The Knowledge Engineering Review, Vol. 28:4, 479–503.
doi:10.1017/S0269888913000088
First published online 19 April 2013

Model-driven development of multiagent systems: a survey and evaluation

GEYLANI KARDAS

International Computer Institute, Ege University, 35100 Bornova, Izmir, Turkey;
e-mail: geylani.kardas@ege.edu.tr

Abstract

To work in a higher abstraction level is of critical importance for the development of multiagent systems (MAS) since it is almost impossible to observe code-level details of such systems due to their internal complexity, distributedness and openness. As one of the promising software development approaches, model-driven development (MDD) aims to change the focus of software development from code to models. This paradigm shift, introduced by the MDD, may also provide the desired abstraction level during the development of MASs. For this reason, MDD of autonomous agents and MASs has been recognized and become one of the research topics in agent-oriented software engineering (AOSE) area. Contributions are mainly based on the model-driven architecture (MDA), which is the most famous and in-use realization of MDD. Within this direction, AOSE researchers define MAS metamodels in various abstraction levels and apply model transformations between the instances of these metamodels in order to provide rapid and efficient implementation of the MASs in various platforms. Reorganization of the existing MAS development methodologies to support model-driven agent development is another emerging research track. In this paper, we give a state of the art survey on above mentioned model-driven MAS development research activities and evaluate the introduced approaches according to five quality criteria we define on model-driven MAS engineering: (1) definition of a platform independent MAS metamodel, (2) model-to-model transformability, (3) model-to-code transformability, (4) support for multiple MAS platforms and finally (5) tool support for software modeling and code generation. Our evaluation has shown that the researchers contributed to the area by providing MDD processes in which design of the MASs are realized at a very high abstraction level and the software for these MASs are developed as a result of the application of a series of model transformations. However, most of the approaches are incapable of supporting multiple MAS environments due to the restricted specifications of their metamodels and model transformations. Also efficiency and practicability of the proposed methodologies are under debate since the amount and quality of the executable MAS components, gained automatically, appear to be not sufficient.

1 Introduction

Design and development of intelligent software agents keep their emphasis on both artificial intelligence and software engineering research areas. In its widely-accepted definition, an agent is an encapsulated computer system (mostly a software system) situated in some environment, and that is capable of flexible autonomous action in this environment in order to meet its design objectives (Wooldridge & Jennings, 1995). These autonomous, responsive and proactive agents have also social ability and interact with other agents and humans in order to complete their own problem solving. They may also behave in a cooperative manner and collaborate with other agents

to solve common problems. To perform their tasks and interact with each other, intelligent agents constitute systems called multiagent systems (MAS).

MAS researchers develop communication languages, interaction protocols and agent architectures that facilitate the development of MASs. They also propose new methodologies (e.g. Gaia (Zambonelli *et al.*, 2003) and Tropos (Bresciani *et al.*, 2004)) and tools for agent-oriented software development because characteristics and challenges of agent-oriented software engineering (AOSE) stretch the limits of current software engineering methodologies as mentioned in Bergenti *et al.* (2004). Besides, AOSE is distinct from object-orientation when agent, goal, role, organization, context and messages are considered as first-class entities. For this reason, various MAS research activities (e.g. Ferber & Gutknecht, 1998; Bernon *et al.*, 2005; Odell *et al.*, 2005) define agent metamodels (model of models) that include these entities and their relations. AOSE researchers also propose new agent modeling languages (e.g. Bauer *et al.*, 2001; Depke *et al.*, 2001; Cervenka *et al.*, 2005) since a system model can be developed by using a proper modeling language, which is used to express the structure of the system by defining a consistent set of rules.

It is obvious that working in a higher abstraction level is of critical importance for the development of MASs since it is almost impossible to observe code-level details of MASs due to their internal complexity, distributedness and openness. As one of the promising software development approaches, model-driven development (MDD) (Selic, 2003) aims to change the focus of software development from code to models, and hence many AOSE researchers believe that this paradigm shift introduced by MDD may also provide the desired abstraction level and simplify the development of complex MAS software. MDD of autonomous agents and MASs has been recognized and become one of the research topics in AOSE area. Contributions are mainly based on the Object Management Group's model-driven architecture (MDA; Object Management Group, 2003), which is the most famous and in-use realization of MDD. Within this direction, some of the AOSE researchers (e.g. Amor *et al.*, 2005; Gracanin *et al.*, 2005; Hahn *et al.*, 2009; Kardas *et al.*, 2009) intend to apply the whole MDD process for MAS development while some of them (e.g. Jayatilleke *et al.*, 2004; Perini & Susi, 2006; Xiao & Greer, 2007) prefer to utilize just MAS metamodels and/or model transformations as needed. Also some researchers (e.g. Pavon *et al.*, 2006; Penserini *et al.*, 2006; Rougemaille *et al.*, 2007) take into account the reorganization of the existing MAS development methodologies to support model-driven agent development. In this paper, we both give a survey of these noteworthy research efforts and an evaluation of the introduced approaches within the scope of model-driven engineering quality.

The survey presents an extensive discussion of the above mentioned model-driven MAS development studies by taking into account the characteristics and methodologies of their proposed approaches, and evaluates the introduced approaches according to five quality criteria we define on model-driven MAS engineering: (1) definition of a platform independent MAS metamodel, (2) model-to-model transformability, (3) model-to-code transformability, (4) support for multiple MAS platforms and finally (5) tool support for software modeling and code generation. In order to provide a comparison between the surveyed model-driven MAS development approaches, the evaluation of each approach according to each criterion is also graded and discussed in this paper.

The remainder of the paper is structured as follows: Section 2 includes a brief introduction of MDD of MAS software for readers who are not familiar with the MDD and related terminology. After this background section, the comprehensive discussion and evaluation of the model-driven MAS development approaches are given in Section 3. Section 4 discusses the results extracted from our evaluation and Section 5 concludes the paper.

2 Model-driven software development

Selic (2003) states that MDD holds promise of being the first true generational leap in software development since the introduction of the compiler. Truthfully, MDD considers the models as the main artifacts of software development and hence its methodology provides a real paradigm shift

in software development. A model can be considered as a set of statements about a software system (Seidewitz, 2003). The validity and appropriateness of models describing software systems are ensured by defining metamodels for those models. AOSE researchers propose various system metamodels (e.g. Ferber & Gutknecht, 1998; FIPA Modeling Technical Committee, 2004; Bernon *et al*., 2005; Molesini *et al*., 2005; Hahn *et al*., 2009) for software agents and many of these metamodels are employed during the MDD of MASs as discussed in the Section 3 of this paper.

A model-driven software development process is based on the definition of (1) domain metamodels for different abstraction levels, (2) mappings between the entities of these metamodels and (3) model transformation rules originating from these entity mappings. According to these predefined transformation rules, a model transformation (Sendall & Kozaczynski, 2003) is automatically applied on instances of these metamodels at run time. In addition to model transformations, model to text transformations are applied for the automatic generation of software codes from system models. Appropriate transformation tools can be used by the software developers during both model to model and model to text transformations.

The most famous and in-use realization of MDD is the Object Management Group's MDA (Object Management Group, 2003). MDA defines several model transformations that are based on the meta-object facility (MOF; Object Management Group, 2002) framework. In MDA, models are first-class artifacts, integrated into the development process through the chain of transformations to coded application. In order to enable this, MDA requires models to be expressed in a MOF-based language. This guarantees that the models can be stored in a MOF-compliant repository, parsed and transformed by MOF-compliant tools, and rendered into XML Metadata Interchange (XMI) for transport over a network (Object Management Group, 2003).

In order to provide a clear distinction between the system design and the underlying architecture and facilitate the implementation of a designed software system in various deployment environments, MDA defines model transformations between three different abstraction layers called *Computation-Independent*, *Platform-Independent* and *Platform-Specific* layers. In each layer, there exists system metamodels which define meta-entities and relations between these meta-entities by taking into consideration the needs of the related abstraction level. Therefore, a metamodel of a software system in the computation independent level is called a *Computation Independent Metamodel (CIMM)*. Likewise, metamodels in platform independent and platform-specific levels are called *Platform-Independent Metamodel (PIMM)* and *Platform-Specific Metamodel (PSMM)*, respectively. Model transformations are defined between these metamodels and those transformations are applied on instance models that conform to these metamodels. As expected, the models of a software system conforming to CIMM, PIMM and PSMM are named as *Computation-Independent Model (CIM)*, *Platform-Independent Model (PIM)* and *Platform-Specific Model (PSM)*, respectively.

A CIM is a view of a system from the computation-independent viewpoint. That means it describes only the business context and business requirements. Software components are not included in a CIM. For instance, a CIM for agent systems may not have any information about agents (e.g. it may just include goals of a system, tasks to be performed and resource dependencies). System requirements can be modeled within a CIM and entities in the CIM can later be used in order to derive agents.

On the other hand, a PIM focuses on the operation of a system while it still hides the details necessary for the implementation of the system in a particular platform. The PIM specifies a degree of platform independency to be suitable for use with a number of different platforms of similar type. A PIM for an MAS can include entities and association of those entities that represent agents, agent roles, goals, agent organizations, interaction protocols, domain knowledge, agent environment, etc. For instance, Amor *et al.* (2005), Pavon *et al.* (2006) and Hahn *et al.* (2009) introduce PIMMs that can be used to produce such PIMs for MASs.

Finally, a PSM includes details of a platform implementation for a specific system design. For example, the exact model of a MAS implemented in JADE agent development framework (Bellifemine *et al*., 2001) can be considered as one of the PSMs of the related MAS. The model

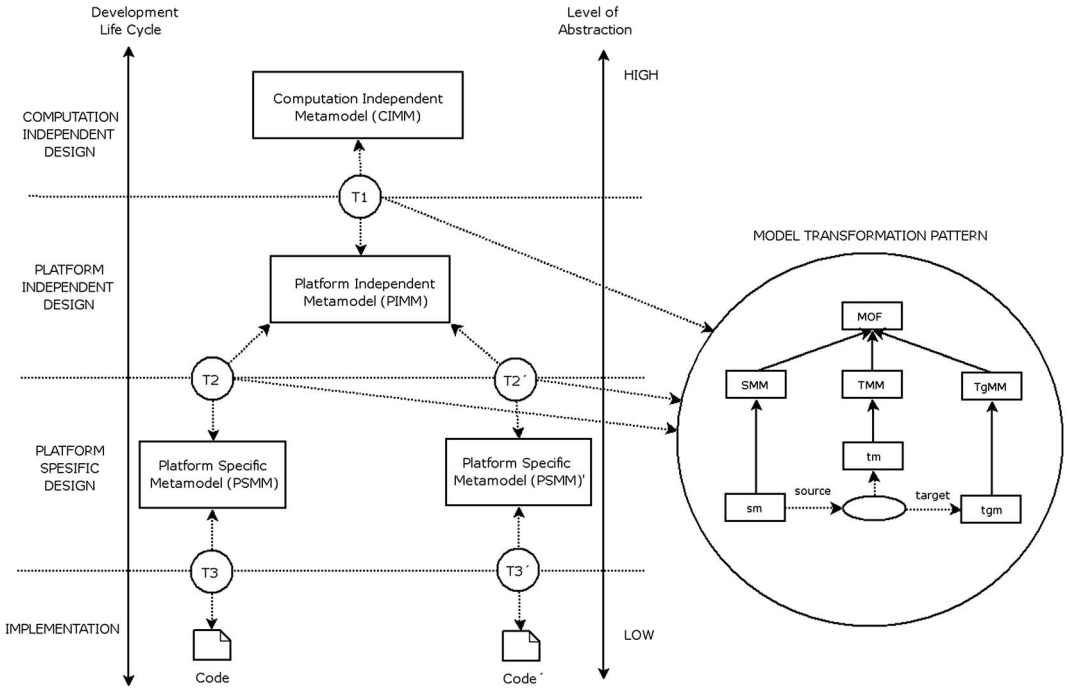

Figure 1 Development process and model transformation mechanism in MDA. The figure represents a slightly modified version of the process described in Kardas *et al.* (2009). MDA = model-driven architecture

presents the implementation details of the MAS and includes the components of the MAS, which are given as instances of the meta-entities defined in the JADE PSMM. Metamodels of various agent development frameworks (e.g. JACK (Agent Oriented Software Group, 2006), NUIN (Dickinson & Wooldridge, 2003) and SEAGENT (Dikenelli *et al.*, 2006)) represent candidate PSMMs that can be used to produce PSMs for MASs.

The development process and the MOF-based transformations between the MDA models are given in Figure 1. The transitions from the computation-independent level to the platform-specific level step by step are specified in the model transformation definitions based on the MOF metamodel. In the transformation pattern depicted in the right-hand side of Figure 1, a source model *sm* is transformed into a target model *tgm*. The transformation is driven by a transformation definition written in a transformation language (e.g. Kalnins *et al.*, 2005; Agrawal *et al.*, 2006; Jouault & Kurtev, 2006). The source and target models and the transformation definition conform to their metamodels *SMM*, *TgMM* and *TMM*, respectively. The metamodel (so meta-metamodel) of these metamodels is MOF. The transformations defined from computation independent to platform independent (T1) and platform independent to platform-specific (T2 and T2′) levels use the CIMM, PIMM and various PSMMs for source and target metamodels in corresponding transformation patterns. After completing the model-to-model transformations according to that pattern, the next and the final step is to provide system implementation by realizing model-to-code transformations (T3 and T3′) for the specific platforms (Kardas *et al.*, 2009).

Since MDA is the most accepted and in-use MDD approach, various approaches exist in AOSE area (e.g. Amor *et al.*, 2005; Gracanin *et al.*, 2005; Xiao & Greer, 2007; Hahn *et al.*, 2009), which aim to provide an MDA for the model-driven MAS development. A broad discussion of these approaches is given in Section 3. For now, it is worth noting that current MDA approaches for MASs mostly do not cover the computation-independent level and generally they are based on developing PIMMs and then converting PIMs to PSMs by model transformations. Bauer and Odell (2005) discuss which aspects of a MAS could be considered at the CIM and the PIM level. Although they do not propose complete CIMs and PIMs, they define conceptual MAS models for the CIM and PIM abstractions. According to Bauer and Odell (2005), a CIM for MASs should

deal with the following aspects: use cases, environment model, domain/ontology model, role model, goal/task model, interaction model, organization/society model and business process models for agent systems. On the other hand, a PIM for MASs should consider the following aspects: interaction protocol model, internal agent model, agent model, service/capability model, acquaintance model and deployment/agent instance model. Bauer and Odell distill these necessary CIM and PIM aspects by summarizing the different approaches of agent-oriented software methodologies and using the Unified Modeling Language (UML) 2.0 (Object Management Group, 2005a) diagrams for agent modeling, which are also discussed in the same paper (Bauer & Odell, 2005).

3 Model-driven development of MASs

The rationale behind MDD and MDA (concrete realization of MDD) discussed in Section 2 has been adopted by various AOSE researchers to define new MAS software development methodologies. A group of researchers (e.g. Gracanin *et al*., 2005; Hahn *et al*., 2009; Kardas *et al*., 2009) adapts the whole MDA process to ease MAS development. They define new MAS metamodels in various abstraction levels, derive model transformation rules from the entity mappings between these metamodels and provide model transformation patterns, which are applied to generate instance MAS models conforming to the proposed metamodels. Another group of researchers (e.g. Jayatilleke *et al*., 2004; Perini & Susi, 2006; Xiao & Greer, 2007) prefers partial application of the traditional MDA process. Model-to-model transformation or just code generation from MAS models are implemented based on requirements. Besides another group of researchers (e.g. Pavon *et al*., 2006; Penserini *et al*., 2006; Rougemaille *et al*., 2007) works on the existing MAS development methodologies, reorganizes them according to the MDD paradigm and produces new versions of these methodologies.

This section presents an extensive survey of the existing model-driven MAS development approaches by taking into account their properties, contributions and limitations (if any). The above discussed preferences of AOSE researchers on MDA application and characteristics of their final work enable us to categorize the approaches in this paper. The MDA application level (full or partial) and origin of the approaches are taken into consideration and hence we group the introduced approaches in three categories: complete MDA implementations, partial MDA implementations and finally reorganized/extended MAS methodologies. Each category and approaches belonging to these categories are discussed in the following Sections 3.2, 3.3 and 3.4, respectively. It is worth noting that some of the surveyed approaches may belong to two different categories. For example, an introduced model-driven MAS development methodology can be an extension of an existing MAS methodology but it may also include a complete application of the MDA during MAS development. We prefer to group such applications by favoring their origins and put them only into the reorganized/extended MAS methodologies category.

During the discussion, the approaches are represented with their abbreviated names (if directly given by the authors) or name of the tools introduced in the papers. If none of them is applicable, a descriptive name is given to represent the related approach.

The survey of the approaches covers a comparative evaluation within the scope of the model-driven engineering quality. Therefore, we first introduce the evaluation criteria used in Section 3.1 and then evaluate each approach according to the given criteria in the proceeding subsections.

3.1 Criteria for the evaluation of the approaches

Currently, a clear methodology for evaluating model-driven software development approaches is unavailable since neither standard evaluation metrics nor formal representation for the effect of human factor in MDD's empirical studies have been determined yet. Recent work in the literature (e.g. Solheim & Neple, 2006; Mohagheghi & Aagedal, 2007) is aimed to define some quality metrics for the evaluation of just metamodeling and/or model transformation instead of a complete evaluation methodology. For instance, Solheim and Neple (2006) propose two-quality

criteria called *Transformability* and *Modifiability*. Transformability means that models must have the ability to be transformed both to other models of greater detail and to executable pieces of code. Modifiability means changes made to the requirements are rendered correctly in the models and reflected in the code. On the other hand, Mohagheghi and Aagedal (2007) define various factors for modeling quality. These factors can be listed as follows: (1) the quality of modeling language(s) used, (2) the quality of tools used for modeling and transformations, (3) the knowledge of the developers for the problem in hand and their experience of modeling languages and tools in use, (4) the quality of the modeling processes used and (5) the quality assurance techniques applied to discover faults or weaknesses. However, their work lacks the definition of metrics for the measurement of these factors.

We think that an alternative for the complete evaluation of an MDD approach can be originated from the comparison of the code-centric and model-centric development processes. Both processes can be used in the development of the same software system and feedback of the developers on the use of models and model transformation processes can be obtained (by interview, filling forms, etc.) and evaluated. Their adaptation and learning tendency for the model-driven approach can be observed and measured. Efficiency and performance advantages of the model-centric approach can also be evaluated according to some predefined metrics.

Unfortunately, most of the model-driven MAS development efforts discussed in this paper do not include such an empirical evaluation of their approaches in their original work. Rare exceptions are MDD4SWAgents (Kardas *et al*., 2009) and CAFnE (Jayatilleke *et al*., 2007). Kardas *et al*. (2009) discuss an empirical evaluation of their approach within its use in a commercial project but the evaluation is very informal (e.g. conversation-based feedback only covering general comments from the users) and evaluation results are not clarified. Evaluation of Jayatilleke *et al*.'s approach is more comprehensive and noteworthy. A group of domain experts (meteorologists) evaluated an MDD toolkit called CAFnE in a simulation of an agent-based meteorology alerting system. Evaluation results are also given in Jayatilleke *et al*. (2007). However, the adequacy of the evaluators is dubious when we consider the facts about the evaluation methodology (e.g. very small number of the evaluators and high-level programming skills of the domain experts). This causes really an important open issue.

For all of the above reasons, an empirical evaluation of the MAS development approaches covered in this paper is obviously impossible and hence it is not considered here. Instead, we have defined five fundamental criteria on capabilities and possessions of the proposed approaches and evaluated the approaches according to those criteria with a three-level grading. Some of those criteria are inherited from the aforementioned model quality work (Solheim & Neple, 2006; Mohagheghi & Aagedal, 2007) and re-construed within the scope of the MAS modeling. These five criteria are listed as follows:

1. *Generic PIMM definition:* provision of a metamodel for platform independent metamodeling of MASs.
2. *M2M Transformability:* inclusion of transformable MAS metamodel(s) and a model-to-model (M2M) transformation mechanism.
3. *M2C Transformability:* inclusion of a model-to-code (M2C) transformation phase for automatic generation of MAS software codes.
4. *Support for multiple platforms:* specification and implementation of M2M (and M2C) transformations for at least two different MAS platforms.
5. *Tool support:* support of software modeling and code generation tools for the MAS developers.

The derivation of the criteria mainly lies in the complete MDD process and model transformation pattern discussed in Section 2. Metamodels and model transformations are apparently inevitable components of the development process. So, definition of the criteria on these components is naturally expected. In addition, we also investigate multiple platform support and code generation capabilities of the approaches. We believe that focusing on just one implementation platform will cause the following question always to remain open: Can the proposed transformations

be applied to generate other PSMs for different MAS platforms? It is therefore important to use at least two PSMMs and apply model transformations between the PIMM and those PIMMs in order to sustain the claim. On the other hand, automatic code generation is also critical. Although software models are the main artifacts of the MDD, they are not sufficient for the real-life implementations of the systems. Definition and application of M2C transformations may provide rapid and error-free production of agent software at least in the template level. Then the developer can complete these auto-generated codes for exact implementation. Hence, M2C transformability is determined as another evaluation criterion. Finally, tool support of the approaches is also taken into consideration since MDD tools such as modeling editors (graphical user interface (GUI) or text based), transformation engines and code generation kits doubtlessly facilitate learning, adoption and use of the proposed methodology by the MAS software developers.

Evaluation of every model-driven MAS development work according to each criterion is graded with one of the following graduation symbols: −, +, ++. The interpretation of each graduation symbol is different for each criterion. Those interpretations are given in Tables 1–5.

Table 1 Grading of the approaches according to criterion: *Generic PIMM definition*

Symbol	Interpretation
−	No inclusion of a PIMM for MASs
+	There is inclusion of a PIMM for MASs but the PIMM is not generic enough to support a variety of MAS platforms. It mostly defines meta-entities of a specific MAS methodology
++	There is inclusion of a generic PIMM for MASs. It is possible to implement MASs modeled according to this PIMM in various MAS platforms

PIMM = Platform-Independent Metamodel; MAS = multiagent system.

Table 2 Grading of the approaches according to criterion: *M2M Transformability*

Symbol	Interpretation
−	Transformation of models is missing
+	There is support for model-to-model transformation to a certain degree. Entity mappings between models or transformation rules are not defined (or they are defined but incomplete). Transformation mechanism is too abstract and not implemented
++	There is support for a complete model-to-model transformation phase in which entity mappings between models and the implementation of the written model transformation rules are all included. Application of the transformations within a case study may also be presented

Table 3 Grading of the approaches according to criterion: *M2C Transformability*

Symbol	Interpretation
−	Generation of software codes from MAS models is missing
+	There is support for model-to-code transformation to a certain degree. Transformations are defined but not implemented or transformations are incomplete and code generation needs too much intervention
++	There is support for an automatic and a complete code generation from models for at least one MAS software development framework. Generated template codes are nearly executable. Application of the transformations within a case study may also be presented

MAS = multiagent system.

Table 4 Grading of the approaches according to criterion: *Support for multiple platforms*

Symbol	Interpretation
−	There is support at most one MAS platform.
+	Definition and implementation of two M2M (and perhaps M2C) transformations for two different MAS platforms
++	There is support for more than two different MAS platforms. Generality of the introduced platform-independent agent metamodel is proved by showing its transformability to at least three different platform specific MAS metamodels

MAS = multiagent system.

Table 5 Grading of the approaches according to criterion: *Tool support*

Symbol	Interpretation
−	Software modeling and code generation tool support for the MAS developers is missing
+	Software tool support to a certain degree. Textual or graphical modeling of MASs is realized only. Extra mechanism is needed for the support of model transformation
++	Various software tools exist for graphical MAS modeling, application of automatic transformation between models and generating MAS software codes from MAS models

MAS = multiagent system.

3.2 Complete MDA implementations

This subsection discusses the approaches that aim to define a model-driven MAS development process based on a complete MDA definition and application. Definition of various metamodels in different abstraction levels, M2M transformations between these metamodels and code generation phases for the exact MAS implementations are all covered within these approaches.

3.2.1 Cougaar MDA

Cougaar MDA introduced in Gracanin *et al.* (2005) aims to improve the productivity of agent system developers by using the MDA approach. The implementation platform is an agent architecture called the Cognitive Agent Architecture (Cougaar), which is based on the human cognitive model of planning. Cougaar MDA attempts to support fully automated generation of software artifacts and simplifies Cougaar-based application development by providing two important abstraction layers. The first layer is the Generic Domain Application Model (GDAM) layer, which represents the PIMM and provides a model of generic agent and domain-specific components found in the domain workflow. The second layer, Generic Cougaar Application Model (GCAM) reflects the PSMM or Cougaar architecture. Cougaar MDA does not include any CIM layer. Application requirements are defined with a Cougaar-specific process definition language and then GDAM components for these requirements are obtained after an automatic transformation. Platform-specific GCAM components are converted to Cougaar/Java codes at the end of the MDA process.

As also stated by the authors, Cougaar is complex and requires considerable mappings and transformations. Application of MDA eliminates this complexity and provides a systematic way of capturing requirements and mapping them from PIMM to PSMM and finally to the code level.

Table 6 shows the evaluation results of the Cougaar MDA approach. The proposed GDAM represents a PIMM for cognitive agent architectures. However, it is too specific for the Cougaar. Hence, Cougaar MDA is graded with (+) for the Generic PIMM definition criterion in our evaluation. There exist some transformations defined between GDAM and GCAM but entity mappings and transformation rules are not discussed in Gracanin *et al.* (2005). That causes

Table 6 Evaluation of Cougaar MDA (Gracanin *et al.*, 2005)

Criterion	Grade
Generic PIMM definition	+
M2M Transformability	+
M2C Transformability	+
Support for multiple platforms	−
Tool support	++

MDA = model-driven architecture; PIMM = Platform-Independent Metamodel.

Cougaar MDA to be graded with (+) for M2M Transformability. It is again graded with (+) for M2C Transformability since transformation mechanism is not clear and exemplified. Also generation of Cougaar/Java code is only mentioned, not discussed. Considering the criterion on support for multiple platforms, Cougaar MDA is graded with (–) because it only supports Cougaar agent platform. On the other hand, a tool with a GUI is provided for modeling Cougaar agent systems and generating Java codes. That causes Courgaar MDA to be fully graded (++) for the tool support criterion.

3.2.2 PIM4Agents

Recently, a group of AOSE researchers from the German Research Center for Artificial Intelligence (DFKI) spent significant effort on creating a PIMM for MASs and using this metamodel in a neatly structured MDA. They first introduce their metamodel and approach in Hahn *et al.* (2006) and Fischer *et al.* (2007), respectively. The improved version of the PIMM, called *PIM4Agents* and the complete MDA based on this PIMM are discussed later in Hahn *et al.* (2009). PIM4Agents groups agent modeling concepts in seven MAS viewpoints called *Multiagent*, *Agent*, *Behavioral*, *Organization*, *Role*, *Interaction* and *Environment*. In Multiagent view, main concepts of a MAS (e.g. Agent, Cooperation, Capability and Interaction) are included. Agent view describes single autonomous entities, the capabilities they have to solve tasks and the roles they play within the MAS. Behavioral view describes agent plans and information flows between control structures of a plan. Cooperation of autonomous agents in a MAS is described in the Organization view. The abstract representations of functional positions belonging to autonomous entities within an organization or any other social relationship are covered by the Role view. Interaction view describes interaction protocols of the agents. Finally, Environment view contains any kind of resource that is dynamically created, shared or used by the agents or organizations.

Since it is not feasible to discuss whole structure of PIM4Agents in here, only the partial PIM4Agents metamodel reflecting the Organizational aspect is described in this paper to give some flavor of the approach. Interested readers may refer to Hahn *et al.* (2009) for the full description of the PIM4Agents.

As depicted in Figure 2, Hahn *et al.* describe the *Cooperation* as a social structure in which *Agents* and *Organizations* can take part. Cooperation binds *Instance* (run-time object of an Agent) to the *DomainRoles* it requires through the concept of a *Binding*. A Cooperation has also its own internal Protocol (through the concept of an *InteractionUse*) that specifies how the members of a Cooperation communicate with each other. For the purpose of interaction, DomainRoles are bound through the concept of an *ActorBinding* to *Actors* that can be considered as representative entities within the corresponding interaction protocols.

Hahn *et al.* define model transformations between PIM4Agents and metamodels of two agent development frameworks (JADE (Bellifemine *et al.*, 2001) and JACK (Agent Oriented Software Group, 2006)) in order to provide MDD of MASs (Hahn *et al.*, 2009). Both metamodels of JADE and JACK are considered as PSMMs and integrated into the proposed MDA. Therefore, both JADE- and JACK-specific counterparts of an instance MAS model conforming to PIM4Agents can be obtained after application of the model transformations defined between PIM4Agents and these PSMMs.

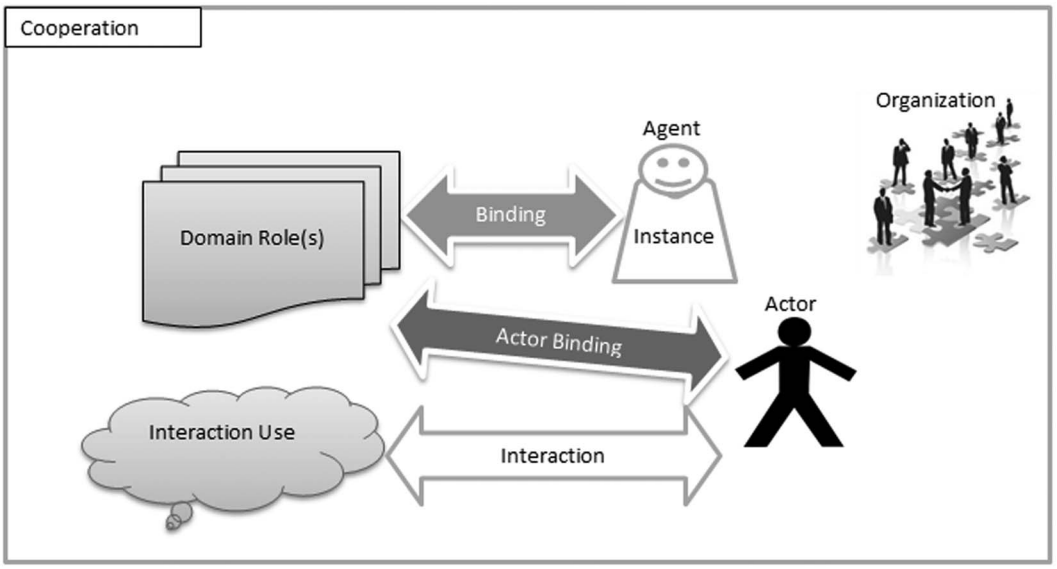

Figure 2 The metamodel reflecting the organizational aspect of PIM4Agents (adapted from Hahn *et al.*, 2009)

Table 7 Evaluation of PIM4Agents (Hahn *et al.*, 2009)

Criterion	Grade
Generic PIMM definition	++
M2M Transformability	++
M2C Transformability	+
Support for multiple platforms	+
Tool support	++

PIMM = Platform-Independent Metamodel.

Apparently, entity mappings between PIM4Agents and JACK metamodel are more complete than the mappings between PIM4Agents and JADE in Hahn *et al.*'s work since design principles of JACK framework and PIM4Agents seem coinciding with each other. However, extensions for the standard JADE framework are required especially when support for agent organizations is considered. For this purpose, two concepts called *Organization* and *Role* are introduced as an extension to the JADE Application Programming Interface (API) in Hahn *et al.* (2009). Inclusion of these new meta-entities to the JADE PIMM supports the entity mappings and model transformations with PIM4Agents. But incompatibility encountered during exact MAS implementation for standard JADE platform and problems in automatic code generation still remain.

Abstracting from existing agent-based metamodels, programming languages and platforms, PIM4Agents presents a PIMM for modeling MASs from various agent viewpoints. Within this context, Hahn *et al.*'s work can be graded with (++) for generic PIMM definition. It also gets (++) for M2M Transformability because model transformations between PIM4Agents and metamodels of two agent development frameworks are given. Also entity mappings and model transformations are discussed and exemplified. Code generation from JADE (Bellifemine *et al.*, 2001) and JACK (Agent Oriented Software Group, 2006) PSMs is considered. But exact implementation of M2C transformability is not discussed. That causes PIM4Agents to get (+) for M2C Transformability. Table 7 lists all grades of PIM4Agents.

Since MDD of MASs in two agent platforms (JADE and JACK) is provided in Hahn *et al.*'s work, PIM4Agent's support for multiple platforms is graded with (+) as expected. A graphical editor is employed in modeling MASs and generating required codes. So, PIM4Agents gets (++) for tool support criterion.

3.2.3 MDD4SWAgents

Another MDA for MAS development is introduced in Kardas *et al*. (2009). The MDD of Semantic Web enabled software agents (shortly *MDD4SWAgents*) is considered in this approach. Software agents collect Web content from diverse sources, process the information and exchange the results on behalf of their human users in the Semantic Web (Berners-Lee *et al*., 2001). Also autonomous agents can evaluate semantic data within these MASs and collaborate with semantically defined entities such as Semantic Web services. The authors claim that the development of such MASs becomes more complex and hard to implement when requirements of the agents in the Semantic Web environment are considered, so they propose an MDA-based MAS development process to facilitate the implementation of such agent systems.

MDD4SWAgents uses an improved version of an agent PIMM, which is first introduced in Kardas *et al*. (2007a). The metamodel is based on a core model discussed in Kardas *et al*. (2006) and provides add-ons for the core model in order to support the use of model in MDD as a PIMM. Major entities of the PIMM are the *Semantic Web Agent*, *Semantic Web Organization*, *Semantic Web Service*, *Ontology* and agent plan entities (e.g. *Semantic Service Register Plan*, *Semantic Service Finder Plan* and *Semantic Service Executor Plan*) for modeling agent behaviors and task executions. A Semantic Web Agent is an autonomous entity which is capable of interaction with both other agents and Semantic Web Services. Semantic Web Agents constitute Semantic Organizations according to their organizational roles. Semantic Web Agents play roles, use ontologies to maintain their internal knowledge and infer the environment based on the known facts.

Kardas *et al*. (2007b) use the above summarized MAS metamodel in a model transformation process for achieving various platform-dependent MAS components. In their most recent work (Kardas *et al*., 2009), they define model transformations between their PIMM and metamodels of two Semantic Web enabled agent development frameworks (SEAGENT (Dikenelli *et al*., 2006) and NUIN (Dickinson & Wooldridge, 2003)) in order to prove applicability of their approach for various agent platforms.

Similar to Hahn *et al*'s. (2009) work, MDD4SWAgents also suffer from the deficiencies of the target agent platforms. As also stated by the authors, model transformations designed for NUIN platform differ from the transformations designed for SEAGENT platform in two major viewpoints: Completeness and Complexity of the transformations. Model transformations from their PIMM to SEAGENT PSMM are more complete and productive than NUIN PSMM. However, new entity definitions are required for NUIN metamodel to complete the exact realization of agent PIMs in NUIN environment especially when we consider Semantic Web service structures. This is because Semantic Web support of NUIN remains only in agent configuration and knowledge store declaration. That metamodel extension in NUIN PSMM is the major drawback of Kardas *et al*.'s work. On the other hand, model transformation for NUIN is more complicated and difficult to implement than SEAGENT transformations. Defined rules for SEAGENT PSMM transformation mostly include attribute settings for the target entities and mapping of source entities into those target entities, while rules written for NUIN transformations include various dynamic model element creations and complex queries for detection of source elements on the pattern. Those differences in model transformations for NUIN and SEAGENT PSMMs are naturally expected because SEAGENT platform and the proposed PIMM are compatible due to their abstractions, design mechanisms and environments that they model. This resembles the compatibility of PIM4Agents and JACK (Agent Oriented Software Group, 2006) and definition of new entities for the JADE (Bellifemine *et al*., 2001) metamodel to support the model transformation in Hahn *et al*'s. (2009) work.

Table 8 lists evaluation scores of MDD4SWAgents. In our evaluation, MDD4SWAgents is graded with (++) both for generic PIMM definition and M2M Transformability criteria. A PIMM for MASs is introduced and model transformations between the proposed PIMM and metamodels of two agent development frameworks are given. Entity mappings and model transformations are discussed and also exemplified within a case study. However, M2CTransformability in MDD4SWAgents gets only (+) because transformations are incomplete and generated

Table 8 Evaluation of MDD4SWAgents (Kardas *et al*., 2009)

Criterion	Grade
Generic PIMM definition	++
M2M Transformability	++
M2C Transformability	+
Support for multiple platforms	+
Tool support	++

PIMM = Platform-Independent Metamodel.

codes need too much intervention at later. MDD of MASs in SEAGENT and NUIN agent platforms are supported in Kardas *et al*.'s work. Hence, support of MDD4SWAgents for multiple platforms is clearly at level (+). On the other hand, MDD4SWAgents's tool support is (++) because various software tools exist for graphical MAS modeling, application of automatic transformation between models and generating software codes from MAS models.

3.3 Partial MDA implementations

We categorize another group of model-driven MAS development approaches as partial MDA implementations. Instead of a general and complete approach, some AOSE researchers prefer to define MAS metamodels for only one abstraction level (e.g. PIM or PSM level) and based on the requirements, they propose either M2M or M2C transformations for system development. This subsection discusses the approaches belonging to this category.

3.3.1 Malaca model

Amor *et al*. (2005) propose an MDA-based MAS development process, which employs a platform-neutral agent model called *Malaca*. Since the direct transition from various MAS development methodologies (e.g. Gaia (Zambonelli *et al*., 2003) and Tropos (Bresciani *et al*., 2004)) to the MAS implementation platforms (e.g. JADE (Bellifemine *et al*., 2001) and JACK (Agent Oriented Software Group, 2006)) is usually too hard and sometimes impossible, Malaca model aims to realize this transition by representing an intermediate agent model which bridges the gap between agent methodologies and implementation platforms. The idea behind the work is to define entity mappings and model transformations between AOSE methodologies and Malaca model and then between Malaca model and agent development platforms. Hence, direct transition can be supplied over these transformations. In order to illustrate the approach, MDA application between Tropos (Bresciani *et al*., 2004) design model and Malaca agent model is discussed within the same paper (Amor *et al*., 2005). Figure 3a illustrates the classic MDA pattern for model transformation while Figure 3b depicts the application of this pattern on the transformation from Tropos design model to Malaca MAS specification.

The contribution of Malaca intermediate model is clear: with just one transformation between an agent-oriented methodology and the Malaca agent model, the resulting MAS could run in various agent platforms. Of course, mapping and transformation between Malaca and the related agent platforms should be defined before this process. However, defined model transformations for the Malaca model can not be automated in situations where the requirements of the design phase are not given in detail. This causes the proposed model to be applicable only for the AOSE methodologies with the detailed design phase. Besides, many of the current AOSE methodologies do not include a standard procedure for automatic model transformation and interpretation of the design phase's diagrams is not available during transformation. This deficiency of the methodologies prevents the exact implementation of the Malaca-based MDA as already stated by the authors (Amor *et al*., 2005).

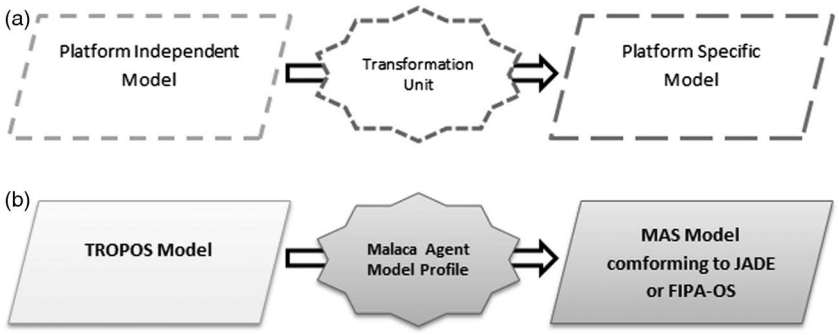

Figure 3 (a) The MDA pattern for model transformation (b) The MDA model transformation from Tropos design model to the Malaca MAS specification using the Malaca Agent Model Profile (adapted from Amor *et al.*, 2005). MDA = model-driven architecture

Table 9 Evaluation of Malaca Model (Amor *et al.*, 2005)

Criterion	Grade
Generic PIMM definition	++
M2M Transformability	+
M2C Transformability	−
Support for multiple platforms	−
Tool support	−

PIMM = Platform-Independent Metamodel.

The Malaca metamodel includes entities and their relations for modeling both internal design of agents and coordination between agents. With above described features of this platform-neutral metamodel, Amor *et al.*'s work is graded with (++) for generic PIMM definition. MDA is applied for the transformation between Tropos design model and Malaca agent model. Entity mappings are discussed in Amor *et al.* (2005), however, transformations based on these mappings usually can not be automated. That causes Malaca model to get (+) for M2M Transformability.

Since only PIM to PSM transformation is proposed, Amor *et al.*'s work gets (−) for M2C Transformability. The conceptual generality of the Malaca model is only tested with Tropos (Bresciani *et al.*, 2004) design model. So, support of Malaca model for multiple platforms is naturally graded with (−). Also Malaca model is not supported with any software modeling and code generation tools for the MAS developers. Grades of Malaca model are shown in Table 9.

3.3.2 CAFnE

The conceptual framework of domain-independent components proposed in Jayatilleke *et al.* (2004) aims to formulate agent systems and modification of agent structures as needed. The overview of the framework is given in Figure 4. Definition of each component type in the framework is denoted with an XML Document Type Definition (DTD) and domain-specific component specifications are given to a transformation module as XML elements conforming to these DTDs. In the transformation module, executable agent codes for each domain component are generated by using Extensible Stylesheet Language Transformation (XSLT) rules and executable binary components. In Jayatilleke *et al.* (2007), the authors also provide a toolkit called *CAFnE (Component Agent Framework for domain Experts)* for their approach in order to make their approach consistent with MDD and use agent models to generate executable codes.

Perhaps the most important contribution of this work to the related research area is the practical evaluation of both the proposed approach and CAFnE toolkit in a real life application. The evaluation considers the MDD of agents employed in a meteorology alerting system. The purpose

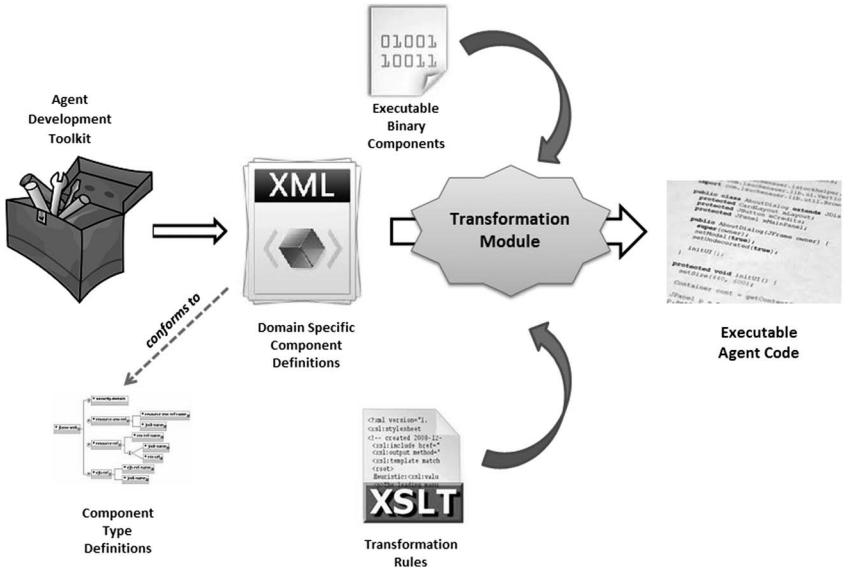

Figure 4 An overview of the conceptual framework of domain independent components (adapted from Jayatilleke *et al.*, 2004)

of the system is to monitor a wide range of meteorological data, alert personnel to anomalous situations (extreme or escalating situations) and so on. The authors claim that domain experts (weather forecasters) with varying programming experience and with no experience with agent design or programming were able to rapidly (35–40 min) become familiar with the CAFnE concepts and begin comprehending an agent system design. They also claim that the users found it easier to work at the higher level of abstraction given by the tool and the overview diagrams provided were seen as useful. Open issues of the work can be listed as the adequacy of the evaluators and simplification of the actual agent-based system. The evaluation of the toolkit and approach was done by five meteorologists and only two of them had no programming experience. Also the evaluation considers the development of a simplified version of the exact agent-based system, which contains some sort of assumptions and data retrieval simulations.

The introduced component model can be accepted as a PIMM for modeling agents, their goals, beliefs and plans. However, this PIMM appears to be too abstract for especially modeling coordination and interaction of agents. Hence, the work is graded with (+) in our evaluation. Transformation of models in different abstraction levels is not considered in the work so it is incapable of M2M Transformability (–). Jayatilleke *et al.* discuss generation of executable agent codes from the DTDs of model elements via XSLT transformation. However, application of the transformations is not clearly illustrated. So, M2C Transformability of CAFnE is graded with (+). Since only JACK agent platform (Agent Oriented Software Group, 2006) is considered, CAFnE's support for multiple platforms is naturally graded with (–). Introduced toolkit provides graphical design of the agent systems based on the proposed component model. But code generation capability of the toolkit is uncertain. Hence, tool support of the work gets (+). All evaluation grades for CAFnE are listed in Table 10.

3.3.3 TAOM4e

Perini and Susi (2006) introduce a model transformation mechanism for agent-based system development. They apply MDA's model transformation pattern in order to obtain UML models from Tropos (Bresciani *et al.*, 2004) MAS structures. Model transformations are designed and implemented according to MOF Query/View/Transformation (QVT) specification (Object Management Group, 2005b). Transformation of a Tropos plan decomposition structure into a UML 2.0 activity diagram is discussed in the paper. Plans in a Tropos plan structure are mapped

Table 10 Evaluation of CAFnE (Jayatilleke *et al.*, 2007)

Criterion	Grade
Generic PIMM definition	+
M2M Transformability	−
M2C Transformability	+
Support for multiple platforms	−
Tool support	+

PIMM = Platform-Independent Metamodel.

Table 11 Evaluation of TAOM4e (Perini & Susi, 2006)

Criterion	Grade
Generic PIMM definition	+
M2M Transformability	++
M2C Transformability	−
Support for multiple platforms	−
Tool support	+

PIMM = Platform-Independent Metamodel.

and transformed into Activity nodes in a UML Activity diagram. Automatic application of the model transformation rules is realized by using a software toolkit called *TAOM4e*, which is also introduced in the same paper. Tropos system models are developed graphically in TAOM4e's GUI and given as an input to the integrated model transformation engine of TAOM4e. Model transformation engine applies the transformation rules on the input model and successfully generates the output model(s).

Perini and Susi's work is a significant effort in application of MDA model transformation pattern in MAS development. Providing a software tool for the real implementation of the transformations is worthwhile. But agent metamodel used during the model transformations is only composed of Tropos structures and the development process is dependent on the Tropos methodology. That may prevent adaptation of the approach into different MAS development methodologies if generality is considered.

In Perini and Susi's work, the metamodel of the Tropos methodology is used as a PIMM during MAS development and a model transformation pattern is applied between Tropos MAS structures and UML constructs. Model transformations are designed and implemented according to MOF QVT specification as discussed above. Hence, TAOM4e gets (+) and (++) for generic PIMM definition and M2M Transformability criteria, respectively (Table 11). The modeler introduced in Perini and Susi (2006) provides graphical modeling of MASs according to the Tropos metamodel. Since code generation capability of the toolkit is uncertain, TAOM4e is graded with (+) for tool support. The proposed approach does not include M2C transformation and it does not support any agent development platforms.

3.3.4 PIM4SOA

Integration of agent systems with service-oriented architectures (SOA) is another emerging agent research track. Agents may constitute the desired composite service architecture and provide the interoperability with other environment resources based on the business process specifications of SOAs. For this purpose, a rapid prototyping framework for SOAs is introduced in Zinnikus *et al.* (2006). The framework is built around an MDD methodology, which can be used for transforming high-level SOA specifications into executable artefacts, both for Web Services and

Table 12 Evaluation of PIM4SOA (Zinnikus *et al.*, 2006)

Criterion	Grade
Generic PIMM definition	−
M2M Transformability	+
M2C Transformability	−
Support for multiple platforms	−
Tool support	+

PIMM = Platform-Independent Metamodel.

Belief-Desire-Intention (BDI; Rao & Georgeff, 1995) agents. The modeling part is the first part of the framework, which is concerned with applying MDD techniques and tools to the design of SOAs. It defines models and transformations that are specific to the concepts used for SOAs, such as Web Service descriptions and plans for autonomous agents. The second part of the framework is the service part that provides a flexible communication platform for Web services. The third part is composed of autonomous agents that deal both with designing and enacting service compositions as well as performing mediation, negotiation and brokering in SOAs.

Zinnikus *et al.* define a PIMM for SOAs called *PIM4SOA* and PSMMs for Business Process Execution Language (BPEL; Andrews *et al.*, 2003) processes and BDI Agents working on the JACK (Agent Oriented Software Group, 2006) environment. The transformation mechanism introduced in that work has similarities with Hahn *et al.* (2006) and Kardas *et al.* (2007b) in the way of defining metamodels, granting mappings and implementing the transformation. However, Zinnikus *et al.* provide a transformation from an agent-free SOA domain to a MAS domain and deals with the agents only in the platform-specific layer. The platform-independent modeling of the MASs is not included in their approach and hence the proposed metamodel cannot be considered as a PIMM for MASs. This causes Zinnkus *et al.*'s work to be graded with (−) for generic PIMM definition. On the other hand, model transformations between PIM4SOA and PSMM of JACK BDI agents are mentioned in Zinnikus *et al.* (2006). But mappings and transformation process are not discussed. So, PIM4SOA gets (+) when we consider M2M Transformability criterion.

Since JACK is the only agent development platform supported and M2C transformation is not taken into consideration, PIM4SOA is graded with (−) both for M2C Transformability and support for multiple platforms. Software modeling tools are utilized during only the specification of agent plans, so tool support of PIM4SOA is at level (+). All grades of PIM4SOA are listed in Table 12.

3.3.5 AMDA

Agent-oriented MDA (AMDA) defined in Xiao and Greer (2007) provides a method for building adaptive MASs with overall development process support. It aims to fill the gap between major AOSE methodologies and agent-oriented development platforms. Behavioral semantics are associated with model constructs and maintained before and after agents translate their behavior to deploy up-to-date requirements. With this objective, the approach of AMDA has similarities with above discussed Malaca (Amor *et al.*, 2005) model (Section 3.3.1). However, AMDA's PIMM is not an intermediate model. The PIMM is more generic and consists of hierarchical business knowledge models and a platform-independent agent model. The authors claim that the domain requirements must be organized in a PIM, in which responsibilities are assignable to conceptual agents and later transformable to a PSM, which agents use to dynamically interpret their behavior while running upon specific platforms. They evaluate their approach in development of an agent-based railway management system. JADE (Bellifemine *et al.*, 2001) platform is chosen for the exact implementation of the system agents. The behavior of the system agents are modeled according to JADE behavior structure since JADE model is considered as a PSMM.

Table 13 Evaluation of AMDA (Xiao & Greer, 2009)

Criterion	Grade
Generic PIMM definition	++
M2M Transformability	++
M2C Transformability	−
Support for multiple platforms	−
Tool support	+

AMDA = agent-oriented MDA.

In fact, AMDA is not just proposed for agent development. It defines a generic agent-based MDA software development methodology instead of the classic object-oriented MDA. This approach is valuable because it supports agent-executable rule-based business models and a high level of abstraction with the direct representation of business requirements. The original AMDA work does not cover the model transformations between various MDA layers. These transformations are exemplified in Xiao and Greer (2009) by taking into consideration software adaptivity. In that work, it can be clearly seen that AMDA use agents as the main constructs of the proposed software model instead of using agents as just target system development components. AMDA's basic metamodel called Adaptive Agent Model is also introduced in Xiao and Greer (2009).

Table 13 shows the evaluation grades of AMDA approach. The above summarized features provides AMDA to get full grade (++) for generic PIMM definition and M2M Transformability. A PIMM for adaptive MASs is introduced and model transformations with entity mappings and rule definitions are given. The proposed approach is also enriched with the use of model configuration and execution tools and gets (+) for tool support. M2C transformation is conceptually mentioned in Xiao and Greer's work. However, exact implementation is not taken into consideration. Also JADE is the only agent development platform supported. These facts cause AMDA to be graded with (−) both for M2C Transformability and multiple platform support.

3.4 Reorganization/extension of existing MAS development methodologies

A group of AOSE researchers reorganizes or extends the existing MAS development methodologies to support model-driven agent development. Originating from the common belief on advantages and facilities of MDD for easy and rapid development of MASs, researchers work on the existing MAS methodologies, produce new versions of these methodologies and provide tool support for the agent development according to these methodologies. This subsection discusses those noteworthy efforts.

3.4.1 IDK

Pavon *et al.* (2006) reformulate their existing agent-oriented methodology called INGENIAS in terms of the MDD paradigm. This reformulation increases the relevance of the model creation, definition and transformation in the context of MASs. New methodology defines a development process, a specification of the results to produce and support tools for modeling and transformation of models. A metamodel for MASs is generated based on the experiences gained by using the INGENIAS methodology in several projects. Modeling and code generation from this metamodel are supported by a software tool called *INGENIAS Development Kit (IDK)*. IDK provides capabilities for model edition, model verification and automatic code generation.

Refined INGENIAS methodology includes two main roles in the development process. As shown in Figure 5a, the *MAS developer* uses the IDK MAS Model Editor to specify MAS models. After these models have been validated, code generation modules facilitate the implementation to deploy in a target platform. When the system has been produced, the testing activities start.

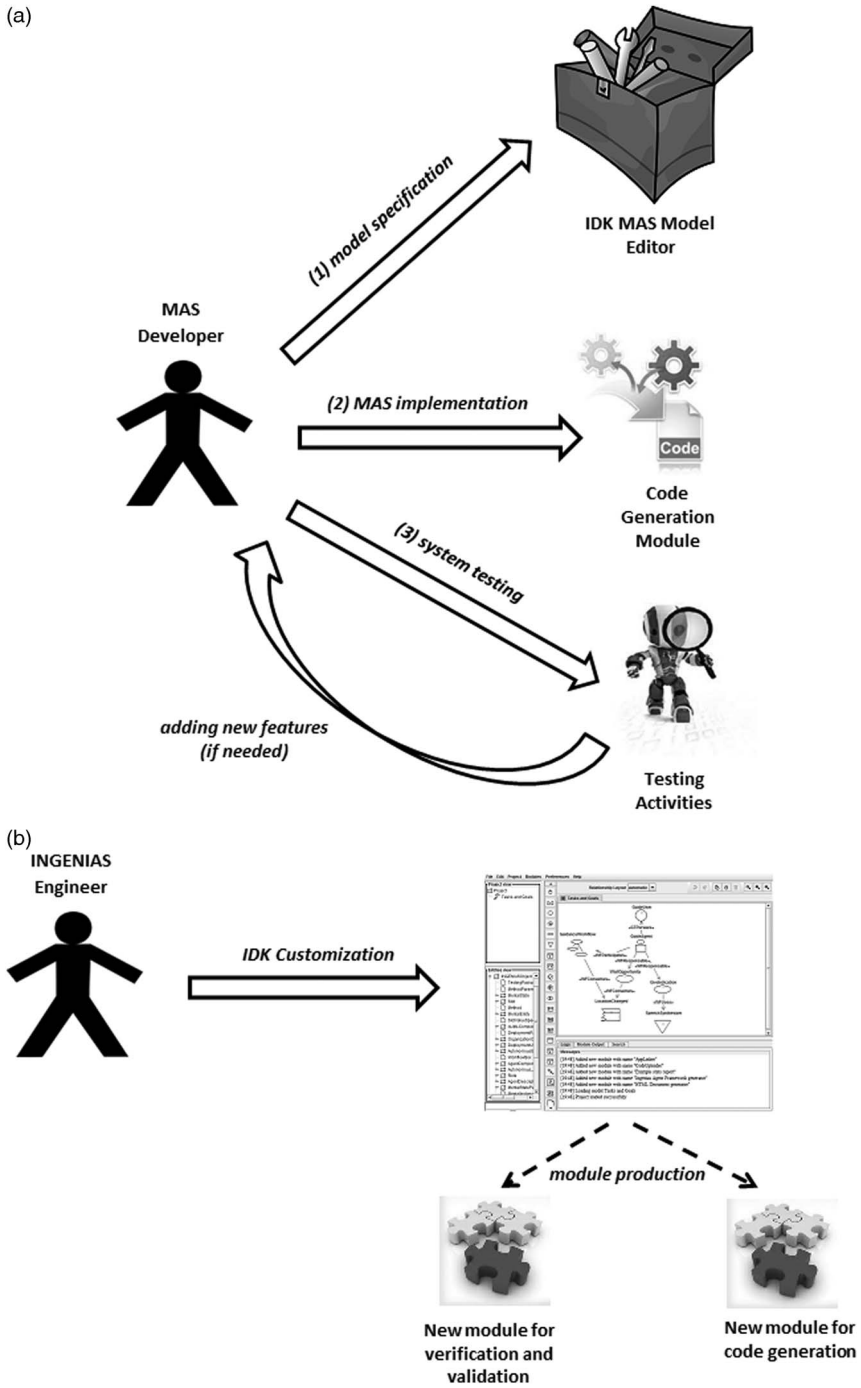

Figure 5 Two main roles in the development process of the refined INGENIAS methodology: (a) MAS developer activities (b) INGENIAS engineer activities (adapted from Pavon *et al.*, 2006). MAS = multiagent systems

The developer may come back to modeling to add new features. On the other hand, *INGENIAS engineer* can customize the editor for a specific purpose by modifying the INGENIAS metamodel if required and produce new modules for verification and validation or for code generation in the target platform (Figure 5b).

Pavon *et al.*'s (2005) work also intends to deal with two issues, which were not addressed by the previous INGENIAS development process. These issues are (1) metamodel evolution for the new requirements and (2) transformation of models into code for different target platforms. The authors claim that it is quite difficult to provide complete metamodels suitable for specific

Table 14 Evaluation of IDK (Pavon *et al*., 2006)

Criterion	Grade
Generic PIMM definition	++
M2M Transformability	−
M2C Transformability	+
Support for multiple platforms	−
Tool support	+

IDK = INGENIAS Development Kit; PIMM = Platform-Independent Metamodel.

platforms and hence they propose a process for partial and incremental transformations that can be driven by the application needs. But it still does not present the complete solution for the first issue, as also stated in their paper. Considering the second issue, the proposed metamodel and transformations described in Pavon *et al*. (2006) support only INGENIAS. Hence, transformation of models into different target platforms (e.g. JADE (Bellifemine *et al*., 2001)) remains uncertain.

Reformulated model-driven INGENIAS methodology includes MAS modeling conforming to a generic PIMM, which is structured in five packages called Agents, Organizations, Goals/Tasks, Interactions and Environment. Within this context, IDK gets (++) for generic PIMM definition (Table 14). Code generation from INGENIAS models is also discussed in Pavon *et al*. (2006). Model-driven process of the IDK is mostly based on the code generation from MAS models. However, implementation of the transformation is not given. That causes IDK to be graded with (+) for M2C Transformability.

Transformation of models in different abstraction levels is not included and none of the agent development platforms is supported. For this reason, IDK gets (−) both for M2M Transformability and support for multiple platforms. On the other hand, IDK MAS Model Editor provides graphical modeling of MASs according to INGENIAS metamodel. Since IDK has not any model transformation capability, tool support of IDK remains at level (+; Table 14).

3.4.2 Model-driven tropos

Another MAS methodology revision according to the MDD paradigm is discussed in Penserini *et al*. (2006). Penserini *et al*. mainly focus on requirements traceability and automated code generation for MAS development. Ideas and standards from MDA are adopted in refining the modeling process algorithm and building tools of Tropos (Bresciani *et al*., 2004) methodology. Tropos capability definitions are revised in order to track whole Tropos development process from early and late requirements definition phases to detailed MAS design and implementation phases. The conceptual agent model of Tropos is evaluated as a PIMM and transformations from this PIMM to JADE (Bellifemine *et al*., 2001) metamodel are defined within the work. In addition to MDA-based model transformations, Penserini *et al*.'s work provides an agent architecture and interaction protocols complying with the IEEE FIPA specifications (IEEE FIPA, 2002) and uses Agent UML (AUML; Bauer *et al*., 2001) for activity and interaction diagrams.

The conceptual agent model of the Tropos methodology is evaluated as a PIMM in Penserini *et al*.'s work and hence it gets (+) for generic PIMM definition. Model transformations between PIM and PSMs are discussed with their transformation rules and examples. M2M Transformability of the approach is fully graded (++).

Generation of Java codes for JADE agents is mentioned in Penserini *et al*. (2006). The intention is to represent capability of agents in the JADE platform. But implementation details of the M2C transformation are not given and that causes the approach to be graded with (+) for M2C Transformation. A model transformation tool is used for the automatic transformation of Tropos models to JADE agent models. That provides model-driven Tropos to get (+) for tool support. But Penserini *et al*.'s work fails in support for multiple platforms (−) since JADE is the only agent platform taken into account. Table 15 lists all grades of the approach.

Table 15 Evaluation of Model-driven Tropos (Penserini *et al*., 2006)

Criterion	Grade
Generic PIMM definition	+
M2M Transformability	++
M2C Transformability	+
Support for multiple platforms	−
Tool support	+

PIMM = Platform-Independent Metamodel.

Table 16 Evaluation of Model-driven ADELFE (Rougemaille *et al*., 2007)

Criterion	Grade
Generic PIMM definition	+
M2M Transformability	+
M2C Transformability	−
Support for multiple platforms	−
Tool support	−

PIMM = Platform-Independent Metamodel.

3.4.3 Model-driven ADELFE

Similar to Pavon *et al*. (2006) and Penserini *et al*. (2006), the approach presented in Rougemaille *et al*. (2007) aims to add a MDD phase to ADELFE (Bernon *et al*., 2003) methodology according to adaptive MAS paradigm by considering two adaptation levels called functional and operational. The functional level is application dependent and close to the decision process of agents while operational level is related to elementary skills of agents. By using the new methodology, a MAS developer can define the functional adaptation in a model conforming to a specific metamodel. Then a model transformation can be executed on this model according to the entity mappings between metamodel of this model and metamodel of an agent architecture for the operational level.

Rougemaille *et al*.'s effort has merit since they intend to provide a development phase to existing ADELFE (Bernon *et al*., 2003) methodology, which consists only of the first three steps of the self-adaptive MAS design life cycle: Requirements, Analysis and Design Workflows. Definition and application of model transformations would reduce the design duration and the complexity of the task for designers as stated by the authors. Hence, agent developers only focus on the system functional adaptation and the agent definition. They do not care about the operational adaptation because it is automatically handled by the model transformations. However, the proposed methodology is just described conceptually and the development phase, which includes model transformations, is not implemented.

In Table 16, evaluation grades of the proposed MDD approach are listed. The approach can be graded with (+) for generic PIMM definition since an adaptive MAS metamodel (AMAS) for the ADELFE methodology is defined. Moreover, entity mappings between AMAS and metamodel of JavAct architecture (Leriche & Arcangeli, 2007) is discussed. But transformations are conceptual and not implemented. So, model-driven ADELFE gets (+) for M2M Transformability.

Only JavAct agent-based middleware is supported in Rougemaille *et al*.'s work. M2C transformation is conceptually given in Rougemaille *et al*. (2007). However, exact implementation is not taken into consideration. Utilization of some existing model editing and transformation tools is indicated. But exact use of these tools is not described. For these reasons, the approach gets (−) when we consider the remaining criteria: M2C Transformability, support for multiple platforms and tool support.

Table 17 Evaluation of the model-driven MAS development approaches

	Generic PIMM definition	M2M Transformability	M2C Transformability	Support for multiple platforms	Tool support
AMDA (Xiao & Greer, 2009)	++	++	−	−	+
CAFnE (Jayatilleke *et al.*, 2007)	+	−	+	−	+
Cougaar MDA (Gracanin *et al.*, 2005)	+	+	+	−	++
IDK (Pavon *et al.*, 2006)	++	−	+	−	+
Malaca Model (Amor *et al.*, 2005)	++	+	−	−	−
MDD4SWAgents (Kardas *et al.*, 2009)	++	++	+	+	++
Model-driven ADELFE (Rougemaille *et al.*, 2007)	+	+	−	−	−
Model-driven Tropos (Penserini *et al.*, 2006)	+	++	+	−	+
PIM4Agents (Hahn *et al.*, 2009)	++	++	+	+	++
PIM4SOA (Zinnikus *et al.*, 2006)	−	+	−	−	+
TAOM4e (Perini & Susi, 2006)	+	++	−	−	+

MAS = multiagent systems; MDA = MDA = model-driven architecture; AMDA = AMDA = agent-oriented MDA; CAFnE = Component Agent Framework for domain Experts; IDK = INGENIAS Development Kit.

4 Evaluation results

The evaluation of model-driven MAS development approaches according to the defined criteria enables us to understand the current progress of related research and achieve some results of the MDA application on MAS development. Regardless of the categories of the surveyed approaches, we again list the grades of each approach for each evaluation criterion altogether in Table 17.

When we examine the grades, we can conclude that almost all of the studies provide a MAS metamodel and define at least a model transformation employing this metamodel. The majority of the proposed MDD processes also include a code generation phase to implement MASs. Also most of the research activities are supported with software tools, which can be used in MAS modeling and M2M and/or M2C transformations. However, support for multiple agent platforms is clearly a big challenge and only MDD processes of PIM4Agents (Hahn *et al.*, 2009) and MDD4SWAgents (Kardas *et al.*, 2009) define transformations for two different agent platforms. None of the proposed approaches consider the transformability for more than two MAS development frameworks.

Based on the evaluation results, it is hard to declare that we currently have an MDD methodology, which is both complete and practical for MAS development. Cougaar MDA (Gracanin *et al.*, 2005), PIM4Agents (Hahn *et al.*, 2009), MDD4SWAgents (Kardas *et al.*, 2009) and model-driven Tropos (Penserini *et al.*, 2006) can be accepted as complete approaches when we consider the ideal MDD process and its application steps discussed in Section 2. But feasibility of the proposals is under debate since none of the transformation patterns owned by these approaches support sufficient agent implementation environments.

We could argue that the lack of a generic and a widely accepted PIMM for MASs is the main reason of the above situation. Such a PIMM can be derived as a combination of already existing MAS metamodels (e.g. Bernon *et al.*, 2005 is a good example) or can be proposed as a brand new metamodel such as PIM4Agents (Hahn *et al.*, 2009). PIM4Agents seems a strong candidate for a common MAS PIMM. However, generality of PIM4Agents is unclear since it is too new and its transformability has been proved only for two MAS platforms. In fact, given transformations from PIM4Agents are not complete and productive as previously discussed in Section 3.2.2. Platform-specific MAS metamodels need extensions for an adequate transformation when we

employ PIM4Agents. Features like support on agent–service interaction, agent adaptivity and various agent planning mechanisms should also be included in PIM4Agents.

On the other hand, a metamodel for MASs should not be too abstract (e.g. Agent Class Superstructure Metamodel (FIPA Modeling Technical Committee, 2004)) in order to be used as a PIMM in model-driven MAS development. Model transformations from the metamodel would be very inefficient and output models of the transformations would be useless if the PIMM is too abstract. Hence, the degree of PIMM generality is critical. It should be both generic enough to support various MAS platforms and concrete enough for productive model transformations. When we take into account the broad diversity relating to purpose, organization and architecture of MASs, it is almost impossible to provide just one MAS PIMM with enough generality and concreteness and use it during MDD of all agent systems.

The above mentioned diversities have caused AOSE researchers to produce specific multiple metamodels for MASs, each with its own uses, and employ them during their model-driven approaches. But originating from specific metamodels naturally makes these approaches impractical and incapable of supporting more than one MAS platform. Evaluation grades on multiple platform support, listed in Table 17, clearly justify this situation.

As a result, we can conclude that the derivation of a general PIMM needed in MDD of all MASs is almost impossible. In addition, utilization of the existing metamodels produce model-driven MAS development methodologies that are weak on multiple platform support. At this point, we may propose a general platform independent modeling approach, which covers the utilization of the existing MAS metamodels; such that we may define *horizontal transformations* between these metamodels and apply these transformations between the instance models of these metamodels in order to provide implementation of the same MAS in various MAS platforms. Referring back to the discussion in Section 2 about the MDD process, we can see that the traditional way is to define metamodels at different abstraction levels and provide vertical model transformations between these metamodels (Figure 1). Mens and Van Gorp (2006) state that refinement of a software system model can be considered as a vertical transformation example in which a specification is gradually refined into a full-fledged implementation, by means of successive refinement steps that add more concrete details. That exactly describes the current research direction followed by most of the AOSE researchers in MDD of MAS. The approaches we evaluate in this paper consider the refinement of an abstract MAS model by applying vertical transformations to receive the most concrete and executable system components. However, it is also possible to define model transformations between the metamodels residing at the same abstraction level. We call these transformations horizontal transformations in which refactoring or a migration of a software system typically takes place (Mens & Van Gorp, 2006).

Design and application of horizontal transformations between MAS metamodels may be encouraging for model-driven MAS development since it provides a general PIMM perspective based on the utilization of the already existing MAS metamodels that are acquired as the result of AOSE community's significant effort. However, the practicability of this approach definitely needs further investigation. Future work on this topic should consider questions such as: What is the cost of such a design? Should we always define transformations in a mesh structure (from all PIMMs to remaining PIMMs) or is just one-to-one transformation enough? Do entity mappings between PIMMs provide an efficient transformation? Is auto-generated PIM satisfactory? To what extent should developers intervene in PIM generation? What is the proportion of the auto-generated model entities (gained as the result of horizontal transformation) among all required MAS PIM entities? Particularly, the determination of the correct structure of the transformations and completeness of the auto-generated MAS PIMs are critical for an appropriate MDD process based on the approach described above.

The last issue that we have considered during the evaluation is the fruitful application of model-driven approaches in MAS design. Although the main artifacts of MDD are output models, software codes for the exact system implementation should eventually be obtained. Within this context, one of the promises of the MDA (in-use realization of MDD) is a faster implementation.

It is stated that for an application development, software developers need to write only 10% of the required code manually, 40% of the code is semi-generic and 50% is generic plumbing code (Herst, 2005). Hence, developers may focus exclusively on the custom code by applying MDA. Unfortunately, the validity of MDA's promise on faster implementation for MAS development is doubtful at present. As discussed above, many of the current model-driven MAS development approaches (e.g. Cougaar MDA (Gracanin *et al*., 2005), PIM4Agents (Hahn *et al*., 2009), MDD4SWAgents (Kardas *et al*., 2009), CAFnE (Jayatilleke *et al*., 2007)) include automatic code generation for various MAS software development frameworks. However, the quantity of the generated code is not satisfactory. For instance, codes required for the agent interactions and internal behavior (plan) structure of agents could not be generated or generated only at the template level in most situations. In order to be executable, vast amount of these auto-generated MAS codes need to be completed manually.

5 Conclusion

A state of the art survey on the model-driven MAS development has been given in this paper. MDD approaches for the development of MASs are discussed and model-driven methodologies proposed by the agent community are evaluated according to quality criteria defined on model-driven engineering.

AOSE is a relatively young research field in computer science and the approaches discussed above can be considered as the first attempts within AOSE to define model-driven MAS development. Our evaluation has shown that most of these attempts contributed to the area by providing MDD processes in which design of the MASs are realized at a very high abstraction level and the software for these MASs are developed as a result of the application of a series of M2M and M2C transformations. On the other hand, some of the remaining approaches just included the partial applications in which M2M transformation or code generation from MAS models are implemented based on the requirements. However, we have concluded that most of the proposals are incapable of supporting multiple MAS environments due to the restricted specifications of their metamodels and model transformations. Also efficiency and practicability of the proposed methodologies are under debate since the amount and quality of the executable MAS components, gained automatically, appear to be not sufficient. In particular, automatic generation of the program codes considering agent interactions and behaviors is not at the desired level, and in most situations intervention of the software developers is required. Hence, extensive modification of the codes is needed before production deployment of the system. At this point, we may also conclude that reflecting autonomous and proactive features of agents and their social relations during model-driven MAS development is currently also in a preliminary state and needs further research.

Acknowledgments

This work is funded by The Scientific and Technological Research Council of Turkey (TUBITAK) Electric, Electronic and Informatics Research Group (EEEAG) under grant 109E125. The author also wishes to thank the anonymous reviewers for their accurate comments on the previous versions of the paper. He was able to improve both his work and the paper significantly by taking their critical comments into account.

References

Agent Oriented Software Group 2006. *JACK Intelligent Agents*. Retrieved on May 16, 2011, from `http://www.agent-software.com/`

Agrawal, A., Karsai, G., Neema, S., Shi, F. & Vizhanyo, A. 2006. The design of a language for model transformations. *Software and Systems Modeling* **5**(3), 261–288.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I. & Weerawarana, S. 2003. *Business Process Execution Language for Web Services*

Version 1.1. Retrieved on May 16, 2011, from http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf

Amor, M., Fuentes, L. & Vallecillo, A. 2005. Bridging the gap between agent-oriented design and implementation using MDA. Lecture Notes in Computer Science **3382**, 93–108. Springer.

Bauer, B. & Odell, J. 2005. UML 2.0 and agents: how to build agent-based systems with the new UML standard. *Engineering Applications of Artificial Intelligence* **18**(2), 141–157.

Bauer, B., Muller, J. P. & Odell, J. 2001. Agent UML: a formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering* **11**(3), 207–230.

Bellifemine, F., Poggi, A. & Rimassa, G. 2001. Developing multi-agent systems with a FIPA-compliant Aaent framework. *Software: Practice and Experience* **31**(2), 103–128.

Bergenti, F., Gleizes, M-P. & Zambonelli, F. 2004. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Kluwer Academic Publishers.

Berners-Lee, T., Hendler, J. & Lassila, O. 2001. The Semantic Web. *Scientific American* **284**(5), 34–43.

Bernon, C., Gleizes, M-P., Peyruqueou, S. & Picard, G. 2003. ADELFE: a methodology for adaptive multi-agent systems engineering. Lecture Notes in Artificial Intelligence **2577**, 70–81. Springer.

Bernon, C., Cossentino, M., Gleizes, M-P., Turci, P. & Zambonelli, F. 2005. A study of some multi-agent meta-models. Lecture Notes in Computer Science **3382**, 62–77. Springer.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. 2004. Tropos: an agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems* **8**(3), 203–236.

Cervenka, R., Trencansky, I., Calisti, M. & Greenwood, D. 2005. AML: Agent Modeling Language – toward industry-grade agent-based modeling. Lecture Notes in Computer Science **3382**, 31–46.

Depke, R., Heckel, R. & Küster, J. M. 2001. Agent-oriented modeling with graph transformations. Lecture Notes in Computer Science **1957**, 105–119. Springer.

Dickinson, I. & Wooldridge, M. 2003. Towards Practical Reasoning Agents for the Semantic Web. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, Rosenschein, J. S., Wooldridge, M., Sandholm, T. & Yokoo, M. (eds.). ACM Press, 827–834.

Dikenelli, O., Erdur, R. C., Kardas, G., Gümüs, O., Seylan, I., Gurcan, O., Tiryaki, A. M. & Ekinci, E. E. 2006. Developing multi agent systems on semantic Web environment using SEAGENT platform. Lecture Notes in Artificial Intelligence **3963**, 1–13. Springer.

FIPA Modeling Technical Committee 2004. *Agent Class Superstructure Metamodel*. Retrieved on May 16, 2011, from www.auml.org/auml/documents/CD2-03-10-31.doc

Ferber, J. & Gutknecht, O. 1998. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems*, Demazeau, Y. (ed.). IEEE Computer Society, 128–135.

Fischer, K., Hahn, C. & Madrigal-Mora, C. 2007. Agent-oriented software engineering: a model-driven approach. *International Journal of Agent-Oriented Software Engineering* **1**(3–4), 334–369.

Gracanin, D., Singh, H. L., Bohner, S. A. & Hinchey, M. G. 2005. Model-driven architecture for agent-based systems. Lecture Notes in Artificial Intelligence **3228**, 249–261. Springer.

Hahn, C., Madrigal-Mora, C. & Fischer, K. 2009. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-agent Systems* **18**(2), 239–266.

Hahn, C., Madrigal-Mora, C., Fischer, K., Elvesæter, B., Berre, A. J. & Zinnikus, I. 2006. Meta-models, models, and model transformations: towards interoperable agents. Lecture Notes in Artificial Intelligence **4196**, 123–134.

Herst, D. 2005. *Model-Driven Architecture for J2EE Development: Promise and Practice*. Retrieved on May 16, 2011, from http://www.orlandojug.org/meeting2005-07-28.html

IEEE FIPA 2002 . *IEEE Foundation for Intelligent Physical Agents (FIPA) Specifications*. Retrieved on May 16, 2011, from http://www.fipa.org/specifications/index.html

Jayatilleke, G. B., Padgham, L. & Winikoff, M. 2004. Towards a Component based development framework for agents. Lecture Notes in Artificial Intelligence **3187**, 183–197. Springer.

Jayatilleke, G. B., Padgham, L. & Winikoff, M. 2007. Evaluating a model driven development toolkit for domain experts to modify agent based systems. Lecture Notes in Computer Science **4405**, 190–207. Springer.

Jouault, F. & Kurtev, I. 2006. Transforming models with ATL. Lecture Notes in Computer Science **3844**, 128–138. Springer.

Kalnins, A., Barzdins, J. & Celms, E. 2005. Model transformation language MOLA. Lecture Notes in Computer Science **3599**, 62–76. Springer.

Kardas, G., Goknil, A., Dikenelli, O. & Topaloglu, N. Y. 2006. Metamodeling of Semantic Web enabled multiagent systems. In *Proceedings of the Multiagent Systems and Software Architecture (MASSA), Special Track at Net.ObjectDays—NODe 2006*, 79–86. Erfurt, Germany.

Kardas, G., Goknil, A., Dikenelli, O. & Topaloglu, N. Y. 2007a. Modeling the interaction between Semantic agents and Semantic Web services using MDA approach. Lecture Notes in Artificial Intelligence **4457**, 209–228. Springer.

Kardas, G., Goknil, A., Dikenelli, O. & Topaloglu, N. Y. 2007b. Model transformation for model driven development of Semantic Web enabled multi-agent systems. Lecture Notes in Artificial Intelligence **4687**, 13–24. Springer.

Kardas, G., Goknil, A., Dikenelli, O. & Topaloglu, N. Y. 2009. Model driven development of Semantic Web enabled multi-agent systems. *International Journal of Cooperative Information Systems* **18**(2), 261–308.

Leriche, S. & Arcangeli, J-P. 2007. Adaptive autonomous agent models for open distributed systems. In *Proceedings of the Second International Multi-Conference on Computing in the Global Information Technology (ICCGI 2007)*, Boiou, M., Costa-Requena, J., Thiebaut, D., Popoviciu, C., Tuy, B. & Van de Velde, G. (eds.). IEEE Computer Society, 19–24.

Mens, T. & Van Gorp, P. 2006. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science* **152**, 125–142.

Mohagheghi, P. & Aagedal, J. 2007. Evaluating quality in model-driven engineering. In *Proceedings of the International Workshop on Modeling in Software Engineering (MISE 2007)*, Atlee, J., France, R., Georg, G., Moreira, A., Rumpe, B. & Zschaler, S. (eds.). IEEE Computer Society, 1–6.

Molesini, A., Denti, E. & Omicini, A. 2005. MAS meta-models on test: UML vs. OPM in the SODA case study. Lecture Notes in Artificial Intelligence **3690**, 163–172. Springer.

Object Management Group 2002. *Meta Object Facility (MOF) Specification Version 1.4*. Retrieved on May 16, 2011, from `http://www.omg.org/cgi-bin/doc?formal/02-04-03.pdf`

Object Management Group 2003. *Model Driven Architecture*. Retrieved on May 16, 2011, from `http://www.omg.org/mda/`

Object Management Group 2005a. *UML 2.0 Superstructure Specification*. Retrieved on May 16, 2011, from `http://www.omg.org/spec/UML/2.0/Superstructure/PDF/`

Object Management Group 2005b. *Meta Object Facility 2.0 Query/View/Transformation Specification*. Retrieved on May 16, 2011, from `http://www.omg.org/spec/QVT/`

Odell, J., Nodine, M. & Levy, R. 2005. A metamodel for agents, roles and groups. Lecture Notes in Computer Science **3382**, 78–92. Springer.

Pavon, J., Gomez-Sanz, J. J. & Fuentes, R. 2005. The INGENIAS methodology and tools. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds). Idea Group Publishing, 236–276.

Pavon, J., Gomez-Sanz, J. J. & Fuentes, R. 2006. Model driven development of multi-agent systems. Lecture Notes in Computer Science **4066**, 284–298. Springer.

Penserini, L., Perini, A., Susi, A. & Mylopoulos, J. 2006. From stakeholder intentions to software agent implementations. Lecture Notes in Computer Science **4001**, 465–479.

Perini, A. & Susi, A. 2006. Automating model transformations in agent-oriented modeling. Lecture Notes in Computer Science **3950**, 167–178.

Rao, A. & Georgeff, M. 1995. BDI agents: from theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 312–319, San Francisco, USA.

Rougemaille, S., Migeon, F., Maurel, C. & Gleizes, M-P. 2007. Model driven engineering for designing adaptive multi-agent systems. Lecture Notes in Artificial Intelligence **4995**, 318–332. Springer.

Seidewitz, E. 2003. What models mean. *IEEE Software* **20**, 26–32.

Selic, B. 2003. The pragmatics of model-driven development. *IEEE Software* **20**, 19–25.

Sendall, S. & Kozaczynski, W. 2003. Model transformation—the heart and soul of model-driven software development. *IEEE Software* **20**, 42–45.

Solheim, I. & Neple, T. 2006. Model Quality in the Context of Model-Driven Development. In *Proceedings of the Second International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2006)*, 27–35, Paphos, South Cyprus.

Wooldridge, M. & Jennings, N. R. 1995. Intelligent agents: theory and practice. *The Knowledge Engineering Review* **10**(2), 115–152.

Xiao, L. & Greer, D. 2007. Towards agent-oriented model-driven architecture. *European Journal of Information Systems* **16**(4), 390–406.

Xiao, L. & Greer, D. 2009. Adaptive agent model: software adaptivity using an agent-oriented model-driven architecture. *Information and Software Technology* **51**(1), 109–137.

Zambonelli, F., Jennings, N. R. & Wooldridge, M. 2003. Developing multiagent systems: the Gaia methodology. *ACM Transactions on Software Engineering and Methodologies* **12**(3), 317–370.

Zinnikus, I., Benguria, G., Elvesæter, B., Fischer, K. & Vayssière, J. 2006. A model driven approach to agent-based service-oriented architectures. Lecture Notes in Artificial Intelligence **4196**, 110–122. Springer.