Meta-models, Models, and Model Transformations: Towards Interoperable Agents

Christian Hahn[1], Cristián Madrigal-Mora[1], Klaus Fischer[1], Brian Elvesæter[2], Arne-Jørgen Berre[2], and Ingo Zinnikus[1]

[1] DFKI GmbH, Stuhlsatzenhausweg 3 (Building D 3-2), D-66123 Saarbrücken, Germany
{christian.hahn, cristian.madrigal, klaus.fischer, ingo.zinnikus}@dfki.de
[2] SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway
{brian.elvesater, arne.j.berre}@sintef.no

Abstract. Services provide an universal basis for the integration of applications and processes that are distributed among entities, both within an organization and across organizational borders: This paper presents a model-driven approach to design interoperable agents in service-oriented architectures (SOA). The approach provides a foundation for how to incorporate autonomous agents into a SOA using principles of model-driven development (MDD). It presents a metamodel (AgentMM) for a BDI-agent architecture and relates AgentMM to a platform-independent model for SOAs (PIM4SOA). In this paper we mainly concentrate our discussions on the service and process aspects of SOA and how transformations to agent technology would look like. We argue that this mapping allows the design of generic agent systems in the context of SOAs that are executable in an adaptive and flexible manner.

Keywords: Modeling Agents, Model-Driven Development, Service-Oriented Architectures, Metamodels.

1 Introduction

Model-driven development (MDD) is emerging as the standard practice for developing modern enterprise applications and software systems. MDD frameworks define a model-driven approach to software development in which visual modeling languages are used to integrate the huge diversity of technologies used in the development of software systems. As such the MDD paradigm, i.e., to develop (i) metamodels that describe the concepts and their relationships and (ii) model transformations that map those concepts and relationships from metamodel to metamodel, provides us with a better way of addressing and solving interoperability issues compared to earlier non-modeling approaches [1].

The current state of the art in MDD is much influenced by the ongoing standardization activities around the OMG Model Driven Architecture (MDA) [2]. MDA defines three main abstraction levels to software development: From a top-down perspective, it starts with a computation independent model (CIM), describing the application context and requirements, that is refined to a platform independent model (PIM), which specifies software services and interfaces independent of software technology platforms.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 123–134, 2006.

The PIM is further refined to a set of platform specific models (PSMs) that describes the realization of the software systems with respect to the chosen software technology platforms.

The focus of this paper is on PIM to PSM transformation development, i.e., the basic idea is to define the transformation from a PIM to an agent PSM. For this purpose, a PIM for Service Oriented Architectures (PIM4SOA) [3] and a metamodel for agent technologies (AgentMM) are presented. If PIM4SOA models can actually be transformed into and executed by agent models, agent systems can be built, so that they can really interoperate with competing technologies in SOAs.

The paper is structured as follows: In Section 2, we present the PIM4SOA metamodel, followed by the metamodel for a specific agent architecture (Section 3). In Section 4, we compare the two metamodels and discuss feasible transformations. Section 5 illustrates related work. Finally, Section 6 presents some conclusions.

2 A Metamodel for Service-Oriented Architectures

Services are loosely coupled, dynamically locatable software pieces, which provide a common platform-independent framework that simplifies heterogeneous application integration. An approach based on agent technologies can be an interesting opportunity when executing services, because:

- agents are self-aware and they acquire the awareness of other agents and their attitudes,
- agents are proactive, whereas services are passive until invoked,
- in contrast to services, agents act in an autonomous manner that is required by many Internet applications,
- agents are cooperative and, by forming organizations, they can provide higher-level and more comprehensive services.

Current standards in the domain of services do not provide any of those functionalities [4]. The metamodel for SOAs addresses the conceptual and technological interoperability barrier and aims at defining platform independent modeling language constructs that can be used to design, re-architect and integrate technologies supporting SOA. The introduction of agents will enable us to design SOAs that are more adaptable and flexible, and, thus, better able to cope with changes over time—which is important for supporting interoperability.

In order to support an evolution of the metamodel for SOAs (PIM4SOA) we have defined a small metamodel core and structured it into groups, each focusing on a specific aspect of a SOA. The current version of the PIM4SOA defines modeling concepts that can be used to model four different aspects of SOAs: Services, information, processes and non-functional aspects. In this paper, we mainly concentrate on the service and process aspects, where services are an abstraction and an encapsulation of the functionality provided by an autonomous entity, and processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc. In general, SOAs are formed by components provided by a system or a set of systems to achieve a shared goal.

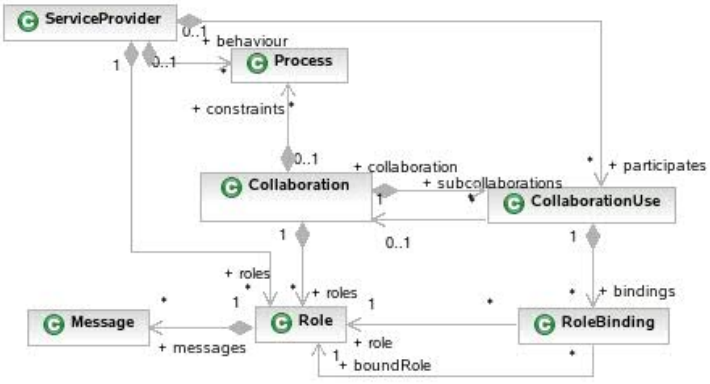

Fig. 1. Service concepts of the PIM4SOA metamodel

The service aspect of the PIM4SOA presents services modeled as collaborations that specify a pattern of interaction between the participating roles. A subset of the metamodel for this aspect is presented in Figure 1. The *Collaboration* specifies the involved roles and their responsibilities. Additionally, a *CollaborationUse* specifies the application of a *Collaboration* in a specific context and includes the *RoleBindings* to entities in that context. Collaborations are composable and the responsibilities of a role in a composite *Collaboration* are defined through *CollaborationUses* by binding roles from the composite to roles of its subcollaborations. The simplest form of *Collaboration* is the binary collaboration, which has no subcollaborations and only two roles; A requester that provides the input, and a provider that produces the output parameters. Therefore, a *Role* represents how a partner participates in the *Collaboration* by providing services and parameters and using services provided by other partners in its service collaboration.

Furthermore, collaborations can have a *Process* that specifies the constraints that define how the involved roles interact. We refer to this as the collaboration protocol. The collaboration protocol is specified from a global view point. Constraints on the expected behavior of a role can be deduced.

The process elements of the PIM4SOA metamodel are shown in Figure 2. This process aspect is closely linked to the service aspect. The primary link is described by the *Process* that belongs to a *ServiceProvider*. The *Process* contains a set of steps (generally tasks), representing the actions to be carried out, and *Interactions/Flows* linking the tasks together. In addition, the *Process* also contains a set of *Flows* between these actions which may indicate the transfer of specific data.

3 A Metamodel for BDI Agents

There are several alternatives when it comes to transforming PIM4SOA models into models that can actually be executed. Agent technologies can provide various

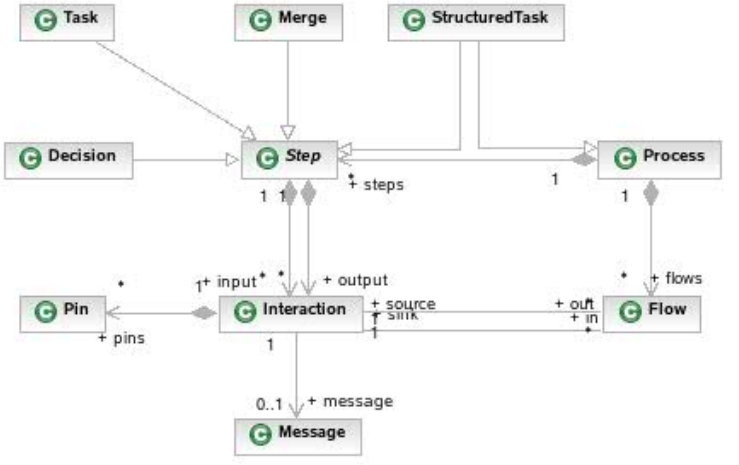

Fig. 2. Process concepts of the PIM4SOA metamodel

contributions to SOAs. For instance, agent technologies allow a flexible and adaptive execution. Also, well-known agent protocols, such as the contract net protocol [5], could be adopted for the service supplier selection, when the number of suppliers is not know at design time. This kind of service selection can be nicely described with agent technologies, while a straightforward model for the PIM4SOA is not available. The integration of agent technologies with PIM4SOAs is therefore a worthwhile enterprise.

For the design of agents with rational and flexible problem solving behavior, the BDI agent architecture has proven to be appropriate during the last decade [6,7]. Three mental attitudes (beliefs, desires, and intentions) allow an agent to act in and to reason about its environment in an effective manner. A vast number of tools and methodologies have been developed to foster the software-based development of BDI agents and multi-agent systems (MAS) [8,9,10,11,12]. Rather than inventing our own agent metamodel, we took a bottom-up approach, by extracting the metamodel (AgentMM) from one of the most sophisticated tools to design BDI agents, namely JACK$^{\mathrm{TM}}$ Intelligent Agents[3] [13]. Figure 3 presents the most interesting part of this metamodel.

Table 1 summarizes the most important high-level BDI concepts. From these, the most relevant one in AgentMM is the concept of *Team*, which can be either atomic, in which case we can refer to it simply as an *Agent*, or a set of required roles—subteams—that all together form the *Team*. It is important to note that it is not necessary for all members of the *Team* to be involved in all the tasks it performs. Rather, for each individual task, a subset of the available roles is selected to actually work on it. The tasks a given *Team* is able to work on are defined by the roles that it is able to fulfill.

[3] JACK$^{\mathrm{TM}}$ is the trademark of an agent oriented model developed by Agent Oriented Software Group. A free evaluation package for JACK Intelligent Agents is available for download.

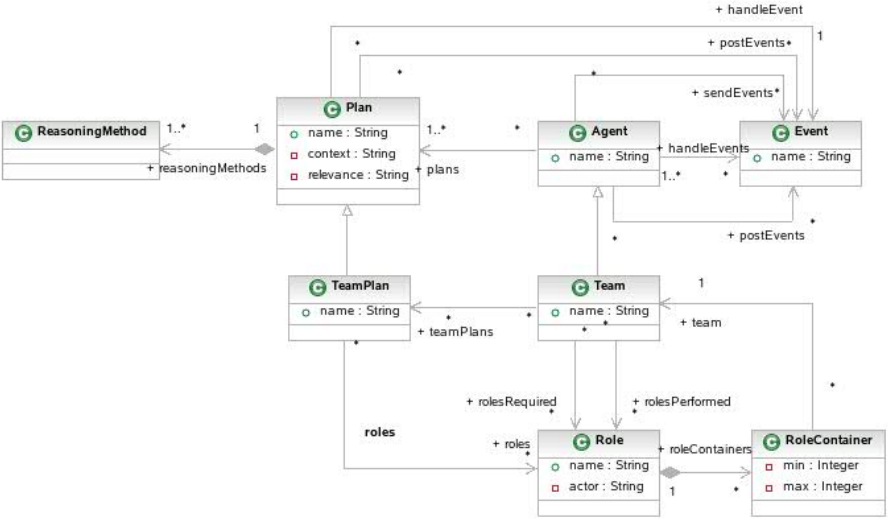

Fig. 3. Partial metamodel for BDI agents

Table 1. High-Level BDI Concepts

High-Level BDI Concepts	
Team	Specifies the structure of one or more entities (Teams/Agents) that is formed to achieve a set of desired objectives
Role	Specifies a role as a type by listing the types of the events the role can deal with
Named Role	An instance of a specific role type. This concepts allows multiple roles with the same type in teams and team plans
Events	The type of stimuli a team, role, or team plan reacts to or posts
Team Plan	Specifies the behavior of a team in reaction to a specific event. In general, a team plan is a set of steps specifying how a particular task is achieved by particular roles
Named Data	Allows the team to store information/beliefs

Table 2. The JACK^{TM} Plan Structure

Structure of Plans	
Triggering Condition	Specification of the event the plan reacts to
Relevance Condition	Additional constraints on the triggering event
Context Condition	Constraints on the state of affairs the plan is designed for to deal with. Usually these constraints access the agents beliefs
Plan Body	Actions that should be taken when the plan becomes active. Can include pure Java code but allows the use of special concepts to drive BDI reasoning

Role definitions are the second most important concept to define teams because a role specifies which messages—in JACK^{TM} those are rather events—the role fillers are able to react to and which messages they are likely to send.

How a team actually reacts to an incoming request is specified by a set of team plans. The structure of a plan can be seen in Table 2. Due to space restrictions, we concentrate in this paper on the concepts of the *Plan Body*—the core part of a plan. Each team plan has an explicitly defined objective (incoming message or internal event) for which this team plan is responsible. When the so-called triggering event is raised and all additional criteria are valid (i.e. *Relevance* and *Context Condition*), a specific team plan is executed by creating an instance of this team plan. As a consequence, a concrete team (already known or newly established) to actually execute the team plan is established.

The process aspect of AgentMM is presented in Figure 4 and it provides the constructs to model the body of the Plans and other, so called, reasoning methods (see Figure 3). The whole package is based on two abstract classes—the *ProcessBase* and *NodeBase*. *NodeBase* represents the basic node in the plan body graph, it provides a reference to its default flow node, the next node in the execution order, and references to the incoming flows, the previous nodes in the execution. *ProcessBase* is the graph container that includes a list of all the nodes and a reference to the starting node. The *Process* class extends *ProcessBase* and represents the main component of the reasoning method body. It is also a process node in itself, which permits modeling nested processes.

Further specialization of the nodes is performed through the *ForkNode* abstract class, which adds an alternative output flow to the *NodeBase*. As shown in Figure 4, the *Node* classes are separated in two groups: The ones that inherit directly from *NodeBase*—only one output flow, and the ones that inherit from *ForkNode*—with the alternative output flow. The semantics of the alternative flow varies depending of the particular node being modeled, for example, in a *DecisionNode* the alternative flow represents the *else* path of the decision, while in a *SubgraphNode* it represents the *fail* path. For detailed semantics of the plan constructs in JACK^{TM} please refer to the JACK^{TM} Documentation [13].

4 Comparison of the Metamodels for PIM4SOA and BDI Agents

In this section, we bring together the metamodel concepts from the two previous sections and relate them to one another in a mapping and a transformation derived from it. Although the concepts from PIM4SOA and AgentMM differ quite significantly the mapping of PIM4SOA models to AgentMM models seems to be feasible as the AgentMM is more expressive.

The first element we inspect is the *ServiceProvider* (presented in Figure 1). At first glance an agent seems to be the best match, but since a *ServiceProvider* references roles, it is recommended to assign it to a team. The name of the *ServiceProvider* coincides with the name of the team and its roles are the roles the team performs.

While a *ServiceProvider* is supposed to represent an atomic team, a *Collaboration* is mapped onto a team that may consist of any number of agents. However, since collaborations do not specify any cardinalities for roles, we can assume that a collaboration asks for exactly one filler for each of the required roles. Correspondingly, we suggest

Table 3. Partial transformations between PIM4SOA and AgentMM concepts

PIM4SOA		**AgentMM**		**Notes**
Concepts	Attributes	Concepts	Attributes	
ServiceProvider	→	***Team***		
	name		name	*name of the service provider is used as team identifier*
	roles		rolesPerformed	*each role is mapped to a role performed by the team*
	behaviour		usesPlans	*for each behavior/process a new teamplan is instantiated*
	participates bindings boundRole		rolesRequired	*roles a team makes use of are specified in the role bindings the service provider participates*
Message	→	***Event***		
	name		name	*name of message is used as event identifier*
Role	→	***Role***		
	name		name	*name of the role (PIM4SOA) is used as role (AgentMM) identifier*
	messages		handleEvents roleContainers min = 1 max = 1	*required roles have a min/max requirement of one filler*
		Team		
	name		name + 'Team'	*name of the role plus the extension "Team" is used as team identifier*
	self		rolesPerformed	*role itself is mapped to the team's performed role*
	→	***TeamPlan***		*TeamPlan responsible for handling service requests*
	name		'Receive' + *name* reasoningMethods body establish pass fail handleEvent postEvents	*bases on Role (PIM4S0A)* *name of role acts as identifier* *body contains send method* *these reasoning methods are automatically generated* *handles service requests* *posts corresponding asynchronous message*
	→	***TeamPlan***		*TeamPlan responsible for invoking services*
	name		'Invoke' + *name* reasoningMethods body establish pass fail handleEvent	*bases on Role (PIM4S0A)* *name of role acts as identifier* *body contains service call* *these reasoning methods are automatically generated* *handles service requests*

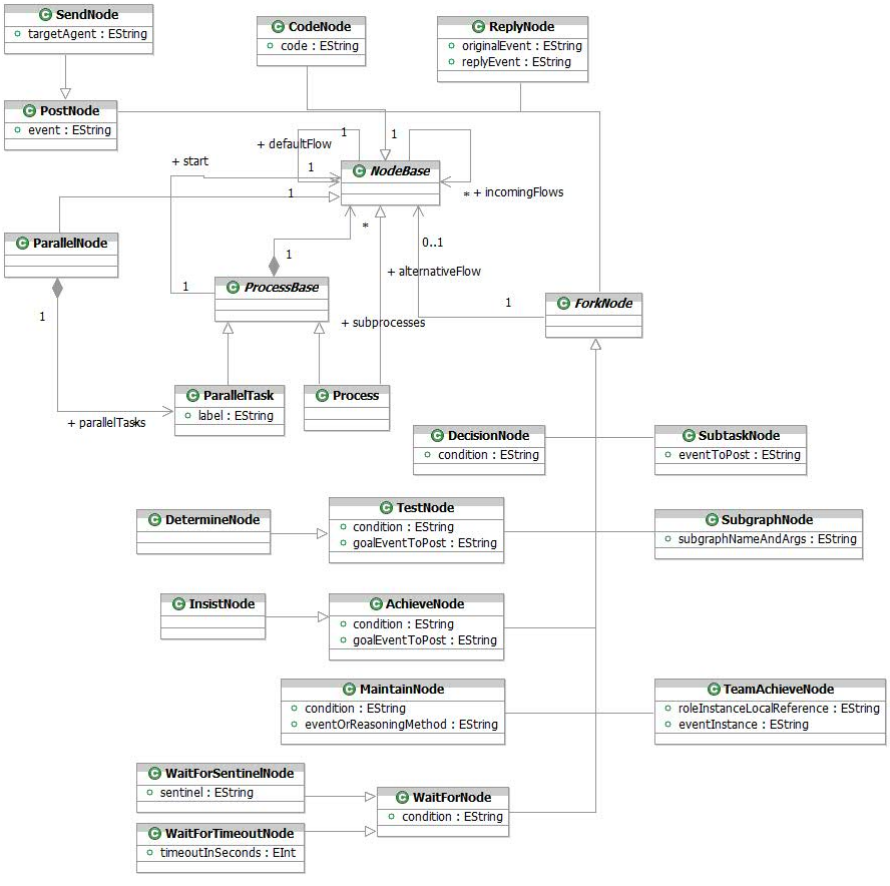

Fig. 4. Process Metamodel of AgentMM (Plan Body)

to map collaborations to team structures where the required roles have a cardinality requirement of exactly one role filler. Although the metamodel for PIM4SOA allows to specify constraints on the behavior of the participating collaborations and their roles, up to now it is unclear how these constraints might look like.

In practical cases, it turns out that a *Process* is provided for *ServiceProviders* and requester roles only. This allows to use the PIM4SOA model in a manner where the information on *Collaborations* and *CollaborationUses* is mapped to a set of teams, where each of the teams consists of a service requester—represented by the team itself—and a set of roles this specific service requester makes use of. Due to the fact that non-composed collaborations in the PIM4SOA are binary, it is always clear who requests and who provides the corresponding service. Interactions other than pure service requests-provisions do not exist. This matches nicely with the fact that, in JACK$^{\text{TM}}$, the interaction of a team as a whole with its attached roles is easy to describe. The behavior of the service requester is therefore mapped to a team plan where each request of a service from a *ServiceProvider* is represented by the construct *team achieve* (see

Table 4. Partial transformations between the process metamodels

PIM4SOA		**AgentMM**		**Notes**
Concepts	Attributes	Concepts	Attributes	
Process	→	***TeamPlan***		*Filter: corresponding Team bases on Service Provider*
			name + 'TeamPlan'	*name := name of service provider*
			reasoningMethods	
	self		body	*the process itself is mapped on the plan body*
	Task		TeamAchieveNode	*transform all tasks with an 'in' Interaction to a TeamAchieveNode*
	Decision		DecisionNode	*one-to-one mapping*
	Merge		ForkNode	*one-to-one mapping*
	in		incomingFlows	*one-to-one mapping*
	out		defaultFlow	*one-to-one mapping*
			establish	*these reasoning methods*
			pass	*are automatically*
			fail	*generated*
	Task		handleEvent	*handle all messages inside a task with an 'out' but no 'in' Interaction*
			postEvents	*post all messages inside a task* with an 'in' Interaction
			roles	*roles a team makes use of are bound in the team plan*

Table 4), that is sent to the respective role in the collaboration the service is involved in (which defines which roles are actually attached to the team that represents the service provider). Thus, the transformation of behavior for service requesters and providers is stereotyped and can be done automatically.

The transformation between the PIM4SOA and AgentMM is finally done using the Atlas Transformation Language (ATL) [14,15] that is a hybrid language designed to express model transformations as described by the MDA$^{\mathrm{TM}}$ approach. The transformation model in ATL is expressed as a set of transformation rules. Tables 3 and 4 illustrate some transformation rules by abstracting from the ATL syntax. The core transformation is described as $ServiceProvider \rightarrow Team$ in Table 3. In this case, for each *ServiceProvider* a *Team* is instantiated that has the same name, performs the same roles and makes use of the roles specified in the *CollaborationUse*, in which it participates, as bound roles. Beside introducing a role in the AgentMM for each role specified in the PIM4SOA, we define a team and two team plans for every role service providers make use of (see $Role \rightarrow Team$ and $Role \rightarrow TeamPlan$ in Table 3). These plans specify how the requested service is invoked and how the corresponding team reacts on a service request.

As discussed before, the process of an PIM4SOA can easily be transformed into team plans (cf. Table 4). As a first approach, we translated the sequential process structure of an PIM4SOA into a sequential team plan.

The final transformation step involves serializing the AgentMM model instance into JACK^{TM} Gcode. This serialization is implemented using MOFScript language [16], which is currently a candidate in the OMG RFP process on MOF Model to Text Transformation. In MOFScript, a set of serialization rules is created following the structure of the source MOF-based metamodel—AgentMM in our case—and the language constructs allow a straight-forward definition of the desired output—Gcode for our purposes.

5 Related Work

This section presents some related contributions in Agent Oriented Modeling, particularly, and Agent Oriented Software Engineering (AOSE) in general.

With regard to modeling languages, the authors of *AgentUML* [11] argue that UML is inappropriate for modeling MAS, so *Agent Interaction Protocols* and *Agent Class Diagrams* are proposed as extensions to UML in order to model these basic features of multiagent systems. Additionally, in [17], the use of *Z* is recommended to overcome the absence of formal semantics of AgentUML.

An additional modeling language approach, the *Agent Modeling Language* (AML) [10], is presented as a semi-formal visual modeling language for the specification of agent systems. However, *AML* does not cover operational semantics since they are considered, by the authors, as a dependency of the specific execution platform.

With respect to methodologies, *Tropos* [9] is an agent-oriented methodology based on the concepts of actor and goal and strongly focused on early requirements. It proposes the use of AgentUML for detailed design and JACK Intelligent Agent^{TM} as implementation platform. In [18], a MDA approach to the transformations in the Tropos methodology is presented.

Prometheus [8] is another known agent-oriented methodology that starts its modeling process by determining the systems percepts and actions. It follows with the definition of *functionalities*. In Prometheus the interaction protocols are modeled using AgentUML and the target implementation platform is BDI-style platforms.

The *Malaca Agent Model* [19] is an approach to Agent Oriented Design using MDA. The *Malaca UML Profile* provides the stereotypes and constraints necessary to create Malaca models on UML modeling tools. In their MDA approach, the transformation is realized from a TROPOS Design Model—as PIM—to a Malaca Model—as PSM.

Related AOSE topics are presented in [12,20,21]. The concept of *Goal-Oriented Interactions* [12,20] presents an interesting way of representing the behavior of agents and provides some additional robustness to the interactions. Cabri et al. propose the *BRAIN Framework* [21] to deal with agent interactions based on the concept of role. In the framework, the description of roles and interactions is realized in an XML notation.

All the mentioned contributions, make valuable points for the specification and modeling tasks in agent systems; however, interoperability among varied agent systems and other technologies is not addressed in these works. Our MDA approach to interoperable agents demonstrates that this objective can be achieved.

6 Conclusions

The paper presented the transformation from PIM to an agent PSM, by explaining how the concepts of a metamodel for SOAs can be transformed to agent related concepts. Therefore an overview of the PIM4SOA, along with its service and process models, was given. Moreover, AgentMM, a metamodel for a specific class of agents, was covered along with its corresponding process model. This transformation that is derived from the mapping of PIM4SOA to AgentMM allows models (i.e., concrete scenarios) that are defined according to the PIM4SOA metamodel to be executed in a flexible, adaptive and generic manner using agent technology.

On one hand, the difference in describing interactions is a challenge when transforming models from PIM4SOA to AgentMM. On the other hand, it also provides chances to actually improve the AgentMM with additional models that are possibly more compact and easier to read.

Our approach shows that interoperability between MAS and other application technologies can be obtained. Further development of the AgentMM could lead to a PIM for agent technologies (PIM4Agents). Since this PIM4Agents would incorporate all relevant high level concepts of the target agent platforms, the interoperability of the generated agent systems would be guaranteed. Moreover, the clear definition of what high level concepts should make part of the PIM4Agents for each particular type of agent technology could prove the greatest contribution of this MDA approach to MAS.

Acknowledgments

The work published in this paper is partly funded by the European Commission through the ATHENA IP (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project) (IST-507849). The work does not represent the view of the European Commission or the ATHENA consortium, and the authors are solely responsible for the paper's content.

References

1. D'Souza, D.: Model-Driven Architecture and Integration - Opportunities and Challenges, Version 1.1, Kineticum. (2001)
2. Object Management Group (OMG): MDA Guide Version 1.0.1, Document omg/03-06-01, June 2003, http://www.omg.org/docs/omg/03-06-01.pdf (2003)
3. Benguria, G., Larrucea, X., Elvesæter, B., Neple, T., Beardsmore, A., Friess, M.: A platform independent model for service oriented architectures. In: Proceedings of I-ESA Conference. (2006)
4. Singh, M., Huhns, M.: Service Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Chichster, West Sussex, UK (2005)
5. Davis, R., Smith, R.G.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence **20** (1983) 63 – 109
6. Rao, A.S., Georgeff, M.P.: Modeling agents within a BDI-architecture. In Fikes, R., Sandewall, E., eds.: KR'91, Cambridge, Mass., Morgan Kaufmann (1991) 473–484

7. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In Lesser, V., ed.: Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, AAAI Press/The MIT Press (1995)
8. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In Giunchiglia, F., Odell, J., Weiß, G., eds.: AOSE. Volume 2585 of Lecture Notes in Computer Science., Springer (2002) 174–185
9. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multiagent Systems **8**(3) (2004)
10. Cervenka, R., Trencanský, I., Calisti, M., Greenwood, D.A.P.: AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling. In: AOSE. (2004) 31–46
11. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. In: AOSE 2000, Springer-Verlag New York, Inc. (2001) 91–103
12. Cheong, C., Winikoff, M.: Hermes: a methodology for goal oriented agent interactions. In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M., eds.: AAMAS, ACM (2005) 1121–1122
13. AOS: JACK Intelligent Agents, The Agent Oriented Software Group (AOS), http://www.agent-software.com/shared/home/ (2006)
14. ATLAS Group, INRIA & LINA, University of Nantes: INRIA, ATL - The Atlas Transformation Language Home Page, http://www.sciences.univ-nantes.fr/lina/atl/ (2006)
15. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: MoDELS 2005, Montego Bay, Jamaica. (2005)
16. SINTEF ICT: MOFScript, http://www.eclipse.org/gmt/mofscript (2006)
17. Huget, M.P.: Modeling Languages for Multiagent Systems. In: AOSE. (2005)
18. Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented modelling. In: AOSE. (2005)
19. Amor, M., Fuentes, L., Vallecillo, A.: Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In: AOSE. (2004) 93–108
20. Cheong, C., Winikoff, M.: Hermes: Designing Goal-Oriented Agent Interactions. In: AOSE. (2005)
21. Cabri, G., Ferrari, L., Leonardi, L.: Supporting the Development of Multi-Agent Interactions via Roles. In: AOSE. (2005)