Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-07-01

A Platform-Independent Model for Agents

Christian Hahn, Cristián Madrigal-Mora and Klaus Fischer

August 2007

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-67608 Kaiserslautern
Tel.: + 49 (631) 205 75-0
Fax: + 49 (631) 205 75-503

Stuhlsatzenhausweg 3
D-66123 Saarbrücken
Tel.: + 49 (681) 302-5151
Fax: + 49 (681) 302-5341

Robert-Hooke-Str. 6
D-28359 Bremen, Germany
Tel.: +49 (421) 218-64 100
Fax: +49 (421) 218-64 150

E-Mail: info@dfki.de WWW: http://www.dfki.de

Deutsches Forschungszentrum für Künstliche Intelligenz
DFKI GmbH
German Research Center for Artificial Intelligence

Founded in 1988, DFKI today is one of the largest nonprofit contract research institutes in the field of innovative software technology based on Artificial Intelligence (AI) methods. DFKI is focusing on the complete cycle of innovation — from world-class basic research and technology development through leading-edge demonstrators and prototypes to product functions and commercialization.

Based in Kaiserslautern, Saarbrücken and Bremen, the German Research Center for Artificial Intelligence ranks among the important "Centers of Excellence" worldwide.

An important element of DFKI's mission is to move innovathons as quickly as possible from the lab into the marketplace. Only by maintaining research projects at the forefront of science can DFKI have the strength to meet its technology transfer goals.

The key directors of DFKI are Prof. Wolfgang Wahlster (CEO) and Dr. Walter Olthoff (CFO).

DFKI's research departments are directed by internationally recognized research scientists:

- ❏ Image Understanding and Pattern Recognition (Director: Prof. T. Breuel)
- ❏ Knowledge Management (Director: Prof. A. Dengel)
- ❏ Deduction and Multiagent Systems (Director: Prof. J. Siekmann)
- ❏ Language Technology (Director: Prof. H. Uszkoreit)
- ❏ Intelligent User Interfaces (Prof. Dr. Dr. h.c. mult. W. Wahlster)
- ❏ Institute for Information Systems at DFKI (Prof. Dr. P. Loos)
- ❏ Robotics (Prof. F. Kirchner)
- ❏ Safe and Secure Cognitive Systems (Prof. B. Krieg-Brückner)

and the associated Center for Human Machine Interaction (Prof. Dr.-Ing. Detlef Zühlke)

In this series, DFKI publishes research reports, technical memos, documents (eg. workshop proceedings), and final project reports. The aim is to make new results, ideas, and software available as quickly as possible.

Prof. Wolfgang Wahlster
Director

A Platform-Independent Model for Agents

Christian Hahn, Cristián Madrigal-Mora and Klaus Fischer

DFKI-RR-07-01

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-012006).

ISSN 0946-008X

A Platform-Independent Model for Agents

Christian Hahn, Cristián Madrigal-Mora and Klaus Fischer

August 29, 2007

Abstract. Various agent-oriented methodologies and metamodels exist to describe multiagent systems (MAS) in an abstract manner. Frequently, these frameworks specialize on particular parts of the MAS and only few works have been invested to derive a common standardization. This limits the impact of agent-related systems in commercial applications. In this paper, we present a metamodel for agent systems that abstract from existing agent-oriented methodologies and platforms and could thus be called platform-independent. This metamodel provides the core language that is used in our agent-oriented software development process that conforms to the principles of Model-Driven Development (MDD). Beside the domain-specific modelling language, we further provide two model transformations that allow to transform the generated models into textual code that can be executed with JACK and JADE.

1 Introduction

Agent-oriented software engineering (AOSE) is rapidly emerging in response to urgent needs in both software engineering and agent-based computing. While these two disciplines co-existed without remarkable interaction until some years ago, today there is rich and fruitful interaction among them and various approaches are available that bring together techniques, concepts and ideas from both sides.

Model-Driven Development (MDD) and Model-Driven Architecture (MDA) as its the most prominent initiative proposed by the Object Management Group (OMG) is a recent trend in the area of software engineering [1]. Our aim is to translate the basic ideas of MDD into methodologies for the design of agent-based systems and in doing so to contribute to bridge the gap between traditional software engineering approach and agent-based system design. To take this one step further, we not only need to integrate MDD into the methodologies of agent-based system design but also demonstrate how such methodologies can be utilized in practical development frameworks for agent-based system design. With respect to our objectives some basic questions arise:

- Agent-oriented methodologies often do not rely on existing agent-based development tools, i.e. they do not provide a straightforward interface for implementation.
- Even if existing methodologies have different advantages when applied to particular problems, usually a unique methodology cannot be applied to each problem without some (minor) level of customization.
- MAS implementation requires deep knowledge regarding technical details of agent architectures, multiagent development tools, and agent concepts.

The question how to fill the gap between agent methodologies and agent-based development tools leads to the development of a framework that (i) standardise the design, (ii) simplifies the implementation of agent systems and (ii) allows to integrate already existing agent frameworks into a single tool box in order to increase the degree of utilization in practice.

In this paper, we show (i) how to build a platform-independent model for agents (PIM4Agents) that abstract from existing agent-based metamodels and platforms and (ii) how MDD can be used to provide a straightforward interface for implementation and thus to simplify the development with agent systems.

The structure of this paper is as follows: Section 2 discusses the very basics of model-driven development. Followed by Section 3 that illustrates related work with respect to modeling languages and agent-based metamodels. Section 4 then defines and illustrates the PIM4Agents which is one of the core parts of our work as it clearly defines the syntax of our modeling language that is defined within this paper. Section 5 and 6 discuss the meatmodels for JACK and JADE, followed by Section 7 that addresses the vertical mappings between the PIM4Agents on the one side and JACK and JADE on the other. In Section 8, a platform-independent model for service-oriented architectures (PIM4SOA) is discussed that serves as base for defining mappings between the PIM4SOA and the PIM4Agents in Section 9. Section 10 addresses the technical realization with respect to model transformations. In Section 11 the main contributions of this paper are discussed followed by Section 12 that concludes this paper.

2 Model-driven Development

MDD is getting more and more important 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 provides us with a better way of addressing and solving interoperability issues compared to earlier non-modeling approaches [2].

The current state of the art in MDD is much influenced by OMG's ongoing standardization activities around the MDA [1]. The MDA approach and its supporting standards allow the realization and integration of one model on multiple platform-specific target models.

Beside the level of abstraction, developing metamodels and model transformations describes an important aspect in MDD. Metamodeling is a controversial topic which is currently critical within OMG's MDA initiative. A metamodel specifies the concepts and their relationships for the purpose of building and interpreting models and thus could be considered as model of a set of models. Metamodels can be developed for describing different domains and different software technology platforms. In its broadest sense, a metamodel is a model of a modeling languages. The term *meta* means transcending or above, emphasizing the fact that a metamodel describes a modeling language at a higher level of abstraction compared to the metamodel itself. To understand the meaning of a metamodel, it is useful to understand the difference between a metamodel and a model. Whilst a metamodel is also a model, a metamodel has two main distinguishing characteristics. Firstly, it must capture the essential features and properties of the language that is being modelled. Thus, a metamodel should be capable of describing a language's concrete syntax. Secondly, a metamodel must be part of a metamodel architecture. All metamodels can be described with a single metamodel, the so-called meta-metamodel, that defines the key to metamodelling as it enables all modelling languages to be described in a unified way. System development is fundamentally based on the use of languages to capture and relate different aspects of the problem domain. The benefit of metamodelling is its ability to describe these languages in a unified way. This means that the languages can uniformly be managed and manipulated and thus tackle the problem of language diversity. Another benefit is the ability to define semantically rich languages that abstract from implementation specific technologies and instead focus on the problem domain at hand. Using metamodels, many different abstractions can be defined and combined to create new languages that are specifically tailored for a particular application domain. As a result, productivity is improved. The Meta Object Facility (MOF) [3] is the common foundation that provides the standard modeling and interchange constructs for defining metamodels and could thus be considered as meta-metamodel.

An important aspect of MDD is the definition of model transformations, which allows automatically transformations of models. A model transformation is a transformation of one or more source models to one or more target models, based on the metamodels of each of these models. In other words the instances of one metamodel are mapped into instances of another metamodel. Such transformations are defined by mapping rules where each of them describes how one, or more elements in the source model should be transformed to the target model. When all mapping rules are applied, the mapping describes the complete transformation from the source model to the target model. Thus, given (i) a source model and (ii) the metamodels of both the source and the target models and applying the defined mappings, the target model could automatically be generated.

MDA defines three main abstraction levels of a system that supports a business-driven approach to software development. From a top-down perspective it starts with a computation independent model (CIM) describing the context and requirements of the software system. The CIM is refined to a platform-independent model (PIM) which specifies software services and interfaces required by the independent software technology platforms. The PIM is further refined to a set of platform-specific models (PSMs) which describes the realization of the software systems with respect to the chosen software technology platforms.

The MOF Query/View/Transformation (QVT) [4] provides a standard specification of a language suitable for querying and transforming models—matching and navigating source elements to initialize target elements—that are represented according to a MOF(-based) metamodel. Basing on source and target metamodels, a model transformation language enables the software developer to match and navigate source elements in order to initialize the target models' elements.

The MDA initiative refers mainly to Object Oriented software development and proved to be effective in relevant application domains. In our ongoing work, we offer a proposal on how to exploit the MDD ideas and techniques in AOSE. Beside the general benefit to improve (i) quality by allowing to reuse models and mappings between models and (ii) software maintainability by favoring a better consistency between models and code, we are especially interested in exploring a framework that (i) establishes interoperability among various agent systems and other information technologies, and (ii) identifies a core metamodel that unifies the most common agent-oriented concepts to increase the efficiency in developing agent-based software applications.

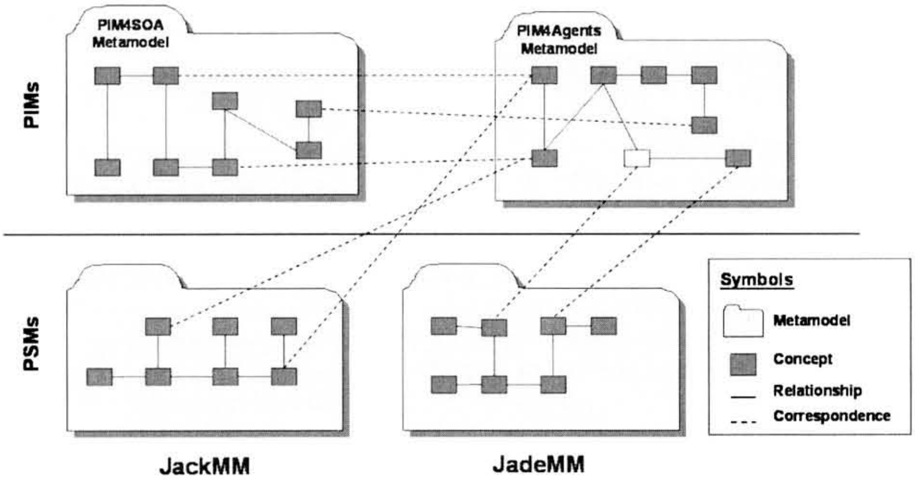

Fig. 1. The overall picture: From a PIM metamodel describing service-oriented architectures (SOA) to a platform-independent model for agents to miscellaneous agent-oriented metamodels.

To increase the interoperability among agent systems, we follow the approach illustrated in Fig. 1. The core part of this framework is a platform-independent metamodel for agents systems (called PIM4Agents) that can be used to model agent system in a very abstract manner without focusing on platform-specific requirements. Basing on the PIM4Agents, we have developed model transformations to various agent specific metamodels on the PSM level that base on agent platforms like for instance the Java Agent DEvelopment Framework [5] or JACK Intelligent Agents [6]. These vertical model transformations allow to transform the abstract models that conforms to the PIM4Agents to concrete code that conforms to the agent-oriented platforms. Beside developing PIM to PSM transformations, we also specified horizontal transformations between a platform-independent metamodel for SOA (called PIM4SOA) and the PIM4Agents to illustrate how MDD can be utilized for the deployment of agents in domain-specific environments like SOA, Peer-to-Peer (P2P) or Grid systems. Furthermore, analyzing the proposed horizontal and vertical transformations allows us to develop a unified metamodel and to decide which concepts should be considered as extensions to meet the domain-specific requirements.

3 Related Work

This section presents some related contributions with respect to agent oriented modelling and MDA approaches in AOSE. We have separated this section into three parts discussing agent modelling languages, agent metamodels, and MDD approaches in AOSE.

3.1 Modelling Languages

Unified Modelling Language (UML) is the de-facto standard industry language for specifying and designing software systems. UML addresses the modelling of architecture and design aspects of software systems by providing language constructs for describing, software components, objects, data, interfaces, interactions, activities etc. UML now provides support for a wide variety of modelling domains, including real-time system modelling and is used more and more in embedded systems.

Agent Modelling Language (AML) is a semi-formal visual modeling language for specifying, modeling and documenting systems that incorporate features drawn from MAS theory ([7]). It is specified as an extension to UML 2.0 in accordance to the OMG's major modeling frameworks (e.g. UML). The ultimate objective of AML is to provide software engineers with a ready-to-use, complete and highly expressive modeling language suitable for the development of commercial software solutions based on multiagent technologies.

Agent UML (AUML) [8] extends UML sequence diagrams to specify agent interaction protocols by providing mechanisms to define agent roles, agent lifelines (interaction threads, which can split into several lifelines and merge at some subsequent points using connectors like AND, OR or XOR), nested and interleaved protocols (patterns of interaction that can be reused with guards and constraints), and extended semantics for UML messages (for instance, to indicate the associated communicative act, and whether messages are synchronous or not). Furthermore, Bauer [9] proposed to extend UML class diagrams to agent class diagrams.

3.2 MAS Metamodels

Aalaadin [10] specifies one of the first developed metamodels for MAS. Based on the three main concepts *Agents*, *Groups* and *Roles*, it takes an organisational-driven (i.e. structural relationship between a set of agents) approach to build MAS. Agents are defined by their role they take on inside an organisation and the capabilities they offer.

Tropos [11] is founded on the idea of using the agent paradigm and related metalistic notions during all phases of the development of software process. Tropos bases on the concepts of actor and goal and strongly focuses on early requirements. It proposes the use of AgentUML for detailed design and JACK Intelligent Agent as implementation platform. As already mentioned, the main concept in Tropos is the concept of an *Actor* that is capable of *Plans* which fulfills a *Goal*, i.e. a *SoftGoal* or *HardGoal* and uses *Resources*. The concept of an *Agent* inherits from *Actor* and may play *Roles*. The *Role* again inherits from the *Actor*.

ADELFE [12] specifies a methodology to develop adaptive MAS by concentrating on cooperative behaviour. The main concept of ADELFE is the *Cooperative Agent* which has *Skills, Aptitudes, Characteristics, Communications*. Furthermore, the agent observes *Cooperation Rules*.

Gaia [13, 14] has been designed to explicitly model and represent the social aspects of open agent systems, with particular attention to the social goals, social tasks or organizational rules. The main concepts of Gaia are *AgentType* which is part of an *Organisation*, collaborates with other *AgentTypes*, provides *Services* and plays several *Roles*. Additionally, a *Role* refers to *Activities*. The roles 'Initiator' and 'Participant' act in a *Communication* that specifies a *Protocol*.

INGENIAS [15] provides both, a methodology and a set of tools to develop agent systems. INGENIAS distinguishes between five viewpoints: organisation viewpoint, agent viewpoint, interaction viewpoint, tasks and goal viewpoint and environment viewpoint. The main concept of INGENIAS is the *Organisation* that contains a *Workflow* and *Group*. A *Workflow* contains *Task* that affects and consumes *MetalEntity* and produces *Interaction*. A *Group* contains again a *Group* and belongs to *Application*, *Resource*, *Agent* and *Role*.

PASSI (Process for Agent Societies Specification and Implementation) [16] is an agent-based methodology to design MAS. The PASSI metamodel [3] is organized in three different domains: Solution domain, agency domain and problem domain. The solution domain covers the concepts *FIPA-Platform Agent*, *Service Description* and *FIPA-Platform Task*. The agency domain covers aspect like *Agent* that has a set of *Roles* that provide a *Service* and solve *Tasks* that includes a set of *Actions*. Furthermore, the *Role* is connected to *Communication* that works on *Agent Interaction Protocols* with a set of *Performatives*. The problem domain contains concepts like *Resource*, *Non Functional Aspects* and *Requirements* that are connected with the *Agent*.

RICA (Role/Interaction/Communicative Action) specifies a metamodel [17] that integrates aspects of Agent Communication Languages (ACL) and organisational models on three different layers: On the first layer, generic concepts of the system (e.g. agent, role and action types) are specified, the second includes social aspects like norms and institutions. The last layer specifies agent interactions via communication.

3.3 Unified MAS Metamodel Proposal

A first attempt towards the development of a unified metamodel was described in [18]. This metamodel was developed by merging the metamodels of ADELFE, Gaia and PASSI and thus combines the strengths of each metamodel. For instance, the unified metamodel covers aspects like (i) cooperative behaviour as described by the ADELFE metamodel, (ii) organisational behaviour as specified by the Gaia metamodel and (iii) FIPA-compliant communication structures as defined by the PASSI metamodel.
A more recent approach towards a unified metamodel was discussed during an AOSE Technical Forum Group meeting in Ljubljana. The attendees agreed on a smaller core part compared to the first draft. In this metamodel, the *Agent* participates in a *Communication* and plays a *Role* that has the ability to solve particular *Tasks*. *Organisations* also refer to *Roles*. The *Cognitive Agent* is a specialisation of *Agent* as it is represented in an *Environment*.

3.4 Agent Platforms

Several platforms already exist to implement agent systems. In the following, we concentrate on JACK[1] and JADE[2].

JACK Intelligent Agents provides programming constructs and concepts for developing complex agent-oriented applications. It bases on the Beliefs, Desires and Intentions model [19] and previous practical implementations of such systems (see [20]). The BDI agent model is an event-driven execution model providing both reactive and proactive behaviour. In this model, an agent has certain beliefs about the environment, desires to achieve, and plans describing

[1] http://www.agentsoftware. com.au/
[2] http://jade.tilab.com/

how to achieve certain activated goals. The BDI architecture is recognised as one of more successfully implemented architecture for developing complex systems in dynamic and error-prone environments (cf. [21]).

JADE (Java Agent DEvelopment Framework) [5] provides programming concepts that simplify the development of MAS as it complies to the FIPA specification by providing the necessary communication infrastructure. In contrast to JACK, it intentionally leaves open the internal agent architecture and necessary concepts. Instead, JADE focuses on communication which is performed through message passing where each agent is equipped with an incoming message box. Standard interaction protocols specified by FIPA such as FIPA-request or FIPA-query can be used as standard templates to build an agent conversation.

3.5 Model-driven Development of MAS

Here we present some of the efforts that have been done to bring Model-Driven Development practices into MAS development.

The Malaca Agent Model [22] 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 modelling tools. In this MDA approach, the transformation is realised from a TROPOS design model—as PIM—to a Malaca Model—as PSM.

Guessoum [23] proposes a MDA-based approach for MAS to fill the gap between existing MAS tools and agent-oriented methodologies and metamodels, respectively. This approach mainly bases on separating the application logic (described in a PIM) from the underlying technology (described in a PSM). Basing on Meta-DIMA, a MDA-based MAS development process defines the PIMs and PSMs by analysing the multiagent applications, defines a library of metamodels by identifying the concepts used and designing the transformation rules to implement a metamodel from its description. A first step has been done by defining a PSM for the multiagent tool DIMA and PIMs from PASSI and Aalaadin/PASSI [24] metamodels.

An update to INGENIAS presented in [25] introduces the *INGENIAS Development Kit (IDK)*, as a way to provide MDD tools for MAS development. It presents the IDK MAS Model Editor, a graphical tool for MAS model creation, and a modular approach to adapt the editor and tools to new metamodels or target platforms. It also proposes that the model generation and metamodel development should be performed in parallel with periodic consistency checks to allow feedback from one activity to the other during the development.

The *Gaia2Jade Process* [26] shows how systems designed following the GAIA methodology, and it corresponding models, can be converted to JADE for deployment. It proposes that the implementation phase should be performed in four stages: communication protocol definition, activities refinement, JADE behavior creation, and agent classes construction. One relevant detail in the behavior creation is that GAIA roles are transformed to 'high level' JADE behaviours, which is a similar approach to the one presented here.

All the previously mentioned contributions in this section, make valuable points for the specification and modelling tasks in agent systems. However, interoperability among varied agent systems and especially among other technologies and domain-specific architectures is not addressed in these works. However, works like [27] and [18] address interoperability within agent systems with completely diverging approaches. On one hand, the *Generic Metamodel* presented in [27] proposes to have a basic, but complete (w.r.t. the concepts that define MAS) metamodel, allowing the generation of systems in different agent platforms. On the other hand, the *Unified Metamodel* [18] presented in Section 3.3 presents some improvements over the original metamodels, but also raises some issues like the complexity of the methodology process to develop systems using it and the construction of tools for it. In the following sections, we address the question of how MDD could contribute to the interoperability between domain-specific architectures and agent platforms with an approach similar to [27] in that we try to set a compact generic metamodel, but within the MDD.

4 Platform Independent Model for Agents

One challenge in defining a platform independent model is to decide which concepts to include and abstract from the target execution platforms (PSMs) that support the architectural style of agent-based systems. Section 3.2 discusses several metamodels for MAS. The only concept all metamodels have in common is the concept of an *Agent*. Some of them also focus on *Role* and *Communication/Interaction*. From this discussion, it is obvious to mention that finding platform-independent concepts for MAS is a complex task. From our point of view, a minimal definition for an agent is that it is an entity that is capable of acting in the environment. It acts in an autonomous manner, i.e. the agent has control over its own behavior and reacts on internal and external stimuli. A further property is the ability to communicate with other agents. Additionally, the agent is capable of perceiving its environment. In the following section, platform-independent concepts and their attributes are discussed that are necessary for designing agents in an adequate manner. In order to support an evolution of this metamodel, it is structured into several aspect each focusing on a specific viewpoint of a MAS.

1. *Agent aspect* describes single autonomous entities, the capabilities they have to solve tasks and their roles they play within the MAS.
2. *Organization aspect* describes how single autonomous entities cooperate within the MAS and how complex organizational structures can be defined.
3. *Interaction aspect* describes how the interaction between autonomous entities or organizations takes place. Each interaction specification includes the actors involved and in which order messages are exchanged between these actors in a protocol-like manner.
4. *Behavioural aspect* describes how plans are composed by complex control structures and simple atomic tasks like sending a message and how information flows between those constructs.

Grouping modeling concepts in this manner allows the metamodel evolution by adding (i) new modeling concepts in the defined aspects, (ii) extending existing modeling concepts in the defined aspects, or (iii) defining new modeling concepts for describing additional aspects of agent systems (e.g. security). In the following, we discuss the four different aspects in more detail and relate each aspect to a small example. This example covers a conference management system (CMS) that has already discussed by several authors (e.g. [28]). We assume that the readers are familiar with the process of submitting a paper to an international conference (e.g. AAMAS). This process starts with a call for papers (CFP) distributed by the program committee (PC). When receiving the CFP, authors decide whether to submit a paper. In case, authors submit their particular paper to the PC that assigns a submission number on it and informs the author about this. After the deadline has passed, the PC distributes all received papers among the PC members that are in charge of providing a review for their assigned papers that is sent back to the PC. Considering all reviews, the PC decides on the accepted papers and sent a message to the corresponding authors to inform them about acceptance or rejection. To keep this example simple, we mainly concentrate on the submission phase in the following.

4.1 Agent Metamodel

Fig. 2 depicts the agent aspect of the PIM4Agents. The metamodel is centered on the concept of Agent, the autonomous entity capable of acting in the environment. An Agent has access to a set of Resources from its surrounding Environment. These Resources may include information or ontologies the Agent has access to. Furthermore, the Agent can perform particular DomainRoles and Behaviours. The DomainRoles are similar to the InteractionRoles specializations of the Role concept that requires a set of Capabilities. Furthermore, the agent may have certain Capabilities that represent the set of Behaviours the Agent can possess. It allows to group Behaviours that, conceptually, have a correspondence with regard to what they allow the Agent to do. Like the Agent, Roles could also refer to Capabilities in order to give it certain patterns of interaction and behavior. Additionally, an Agent could be member in an Organisation that represents the social structure agents can take part in.

Fig. 3 depicts the agent model with respect to our example. In this example we mainly concentrate on the authors' side. We have modeled three agents (i.e. AuthorAgent1, AuthorAgent2

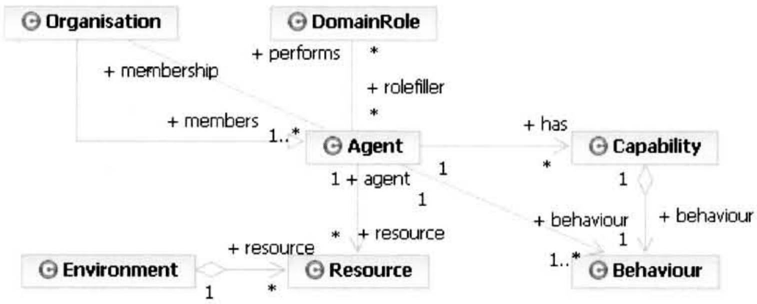

Fig. 2. The metamodel reflecting the agent aspect of the PIM4Agents.

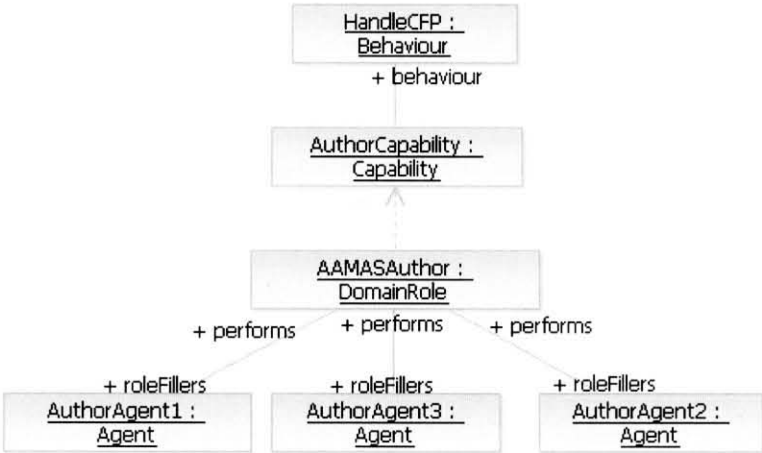

Fig. 3. Agent model of the CMS.

and AuthorAgent3) that all perform the DomainRole AAMASAuthor. This Role has a Capability AuthorCapability that refers to a HandleCFP Behaviour. Details on this behavior are addressed in Section 4.3.

4.2 Organization Metamodel

Fig. 4 depicts the organization aspect of the PIM4Agents. The Organisation is a special kind of Cooperation that also has the same characteristics of an Agent. Therefore, the Organisation can perform Roles and have Capabilities which can be performed by its members, be it Agents or Organisations. The multiple inheritance of the Organisation, from the Agent and the Cooperation, also allows it to have its own internal Protocol that specifies how the Organisation coordinates its members. For the purpose of interaction, DomainRoles are bound to InteractionRoles, where an InteractionRole can be performed by several DomainRoles. This might be important in the case that Protocols are used for different domains.

Fig. 5 depicts an organizational model that conforms to the organizational metamodel. In this example, we modelled the PC as an Organization that requires the InteractionRoles PCChair and PCMember. Furthermore, the Organization PC includes several Agents like PCMemberAgent2 and PCMemberAgent1 that perform the DomainRole AAMASPCMember and PCChair that performs the DomainRole AAMASPCChair. The AAMASPCChair has a Capability that refers to a ReceiveSubmission Behaviour, the AAMASPCMember has a Capability that refers to a Review Behaviour. The DomainRole AAMASPCMember is bound to the InteractionRole PCMember, the DomainRole AAMASPCChair is bound to the InteractionRole PCChair.

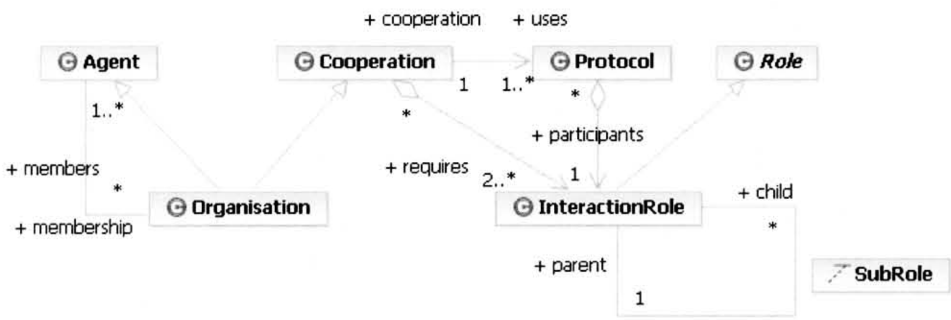

Fig. 4. The metamodel reflecting the organization aspect of the PIM4Agents.

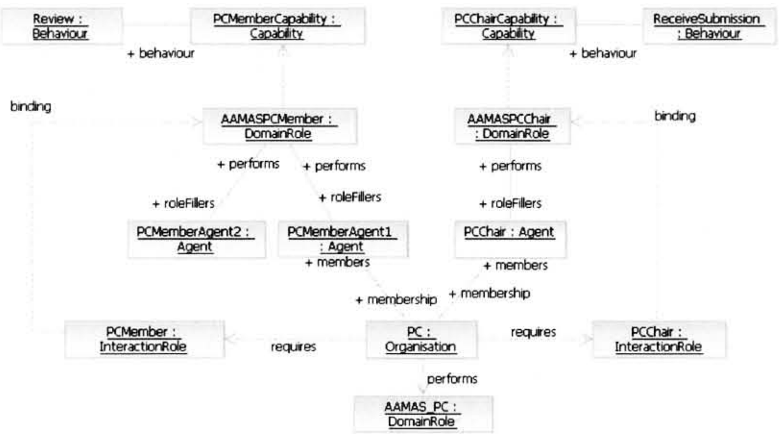

Fig. 5. Organization model of the CMS.

4.3 Behaviour Metamodel

Fig. 6 depicts the behavioral aspect of the PIM4Agents. The Behaviour refers to a set of Flows that could be either of the type InformationFlow or ControlFlow that are contained in the behaviour description. Furthermore, the Behaviour contains a set of Steps that are linked to each other via a Flow. In general, the ControlFlow describes in which order Steps are executed. The InformationFlow describes the order in which information flows between Steps. Each Flow connects exactly two Steps. The concepts StructuredTask and Task are specializations of a Step, i.e. they are again connected by a Flow. A Scope and Plan are further refinements of the StructuredTask. Both are connected to a Condition that mainly defines a set of facts (e.g. boolean values) that are connected by a logical operator. The Plan for instance may refer (i) to a precondition that has to be satisfied in order to execute the Plan and (ii) to a postcondition that defines the state (the fact the should be valid) after the Plan execution. Due to reason how elements in the behavioural viewpoint are structured, Plans could either be composed by more complex control structures (i.e. Scope) or by simple atomic activities (i.e. Task).

The concepts that could be considered as Scope are depicted in Fig. 7. In a first step, we distinguish between the sequential, iterative, and split order of execution. This is reflected by the concepts Sequence, Split and Loop in Fig. 7. The Split is again structured into (i) a Parallel concept that is further partitioned into ANDParallel and XORParallel and (ii) a Decision concept that is further partitioned into ORDecision, XORDecision, and ANDDecision. As a Scope can be considered as specialization of StructuredTask and Step, each Scope can again include sub-scopes to allow the definition of complex control structures.

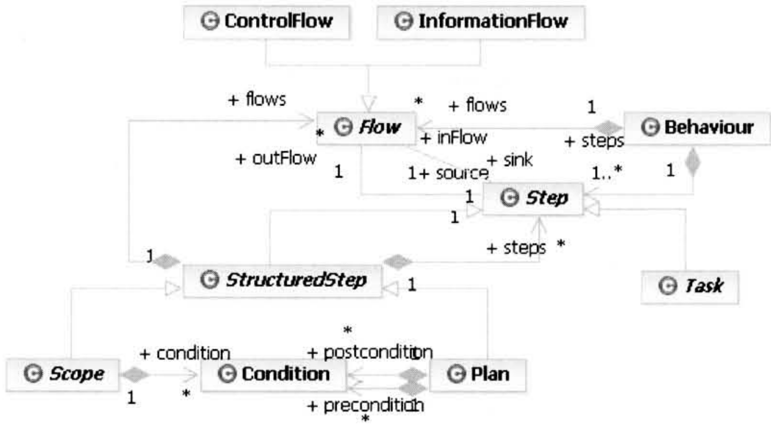

Fig. 6. The metamodel reflecting the behaviour aspect of the PIM4Agents.

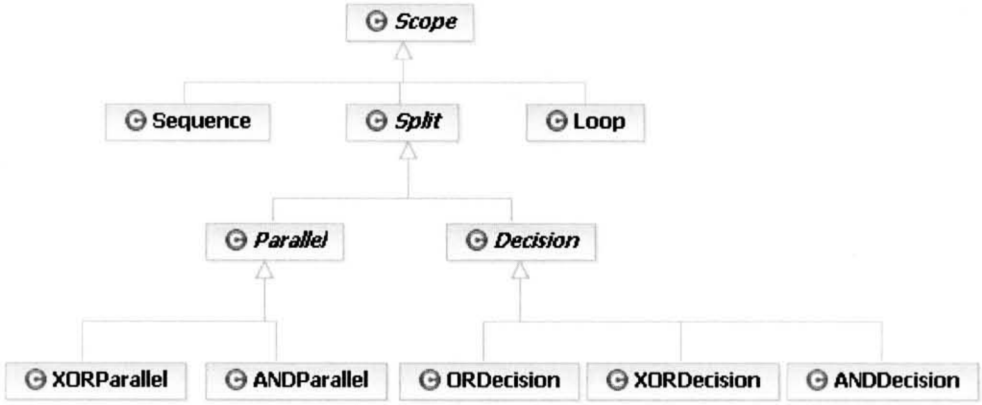

Fig. 7. The specializations of a Scope.

The concepts that could be considered as atomic Tasks are depicted in Fig. 8. Specializations are for instance SendMessage and ReceiveMessage that both refer to a particular Message, InternalTask that could be used to define code or internal statements like the assignment of variables, Wait to express that the Agent/Organization is waiting to meet certain conditions like for instance a time out and InitiateProtocol to start the referred Protocol.

Fig. 9 depicts the HandleCFP Behaviour that was already mentioned in the context of the agent model in Fig. 3. The HandleCFP Behaviour includes one Plan (i.e. HandleCFPPlan) that could in principle be connected with other Plans on this level via the ControlFlowInstance1. For the sake of simplicity, we have not illustrated all ControlFlows in Fig. 9. The HandleCFPPlan includes a Sequence HandleCFPSequence that could again be linked to other control structures on the same level via the ControlFlowInstance2. This Sequence includes two Steps, the ReceiveCFP ReceiveMessage and the XORDecision WritePaper that are connected via the ControlFlowInstance3. The XORDecision refers to a Condition Busy and includes two Steps, an InternalTask Relax (this path is chosen if the author is busy with other work that has to be finished) and a Sequence WritePaperSequence. This Sequence again contains two Steps, an InternalTask WritePaper (stands for the process of writing the paper) and a SubmitPaper SendMessage that refers to a Message SubmitPaper. The ControlFlow ControlFlowIntance4 connects both Steps.

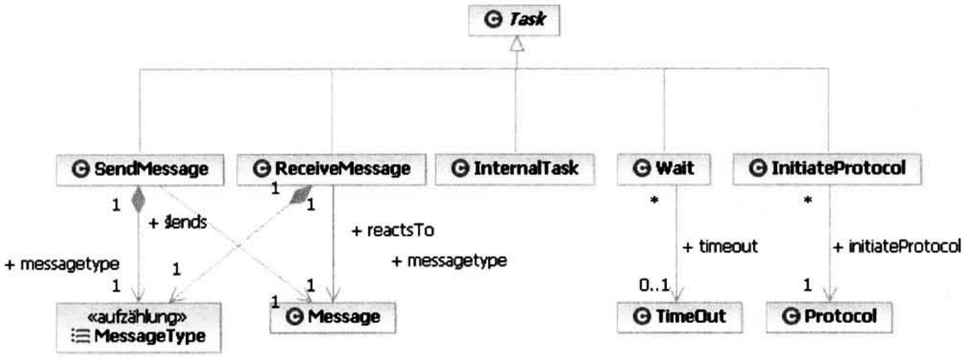

Fig. 8. The specializations of a Task.

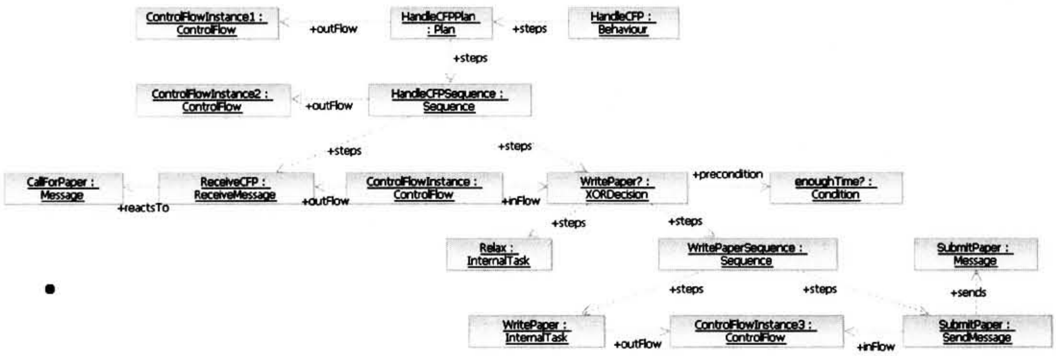

Fig. 9. Behavior model for the HandleCFP Behavior in the CMS.

4.4 Interaction Metamodel

Fig. 10 depicts the interaction aspect of the PIM4Agents. The ability to communicate is one of the core characteristics of agents and group of agents in MAS. A Protocol refers (i) to a set of InteractionRoles (e.g. Buyer and Seller) that interact within the Protocol and (i) to a set of MessageFlows that specify how the exchange of messages is proceed. The InteractionRole can again refer to a set of InteractionRoles as child, meaning that the set of agents that perform the parent InteractionRole is split into the child InteractionRoles. In general, the child InteractionRoles are determined at design time, but filled with the particular agents that perform this role at run time.

A good example why to distinguish between parent and child InteractionRole is the Contract Net Protocol [29] (CNP). In the CNP, the initiator sends in the proposal stage either an `accept-proposal` or a `reject-proposal` to the participant. The decision which message is sent depends on the fact if the participant is considered as best bidder. If this is the case, this participant gets an `accept-proposal`, otherwise a `reject-proposal`. This implicit distinction between best bidder and remaining bidders could be done in the PIM4Agents explicit. The participant would have two children InteractionRoles, i.e. BestBidder and RemainingBidders that are filled at run-time. The MessageFlows again refer to a set of InteractionRoles that are active in the current state, i.e. those Roles that send the specified Messages. Furthermore, it specifies a join and fork operator which are both of the type MessageScope. A MessageScope defines the Messages and their order how these arrive. In particular this means that Messages are connected via a Sequence, Loop, Parallel, OR, XOR, and AND operator. Furthermore, the MessageFlow refers to a TimeOut that specifies the latest point in time a Message should be sent. Beside Messages that can be sent, the MessageFlow may also refer to Protocols that are initiated at some specific point in time in the parent Protocol in order to execute nested Protocols.

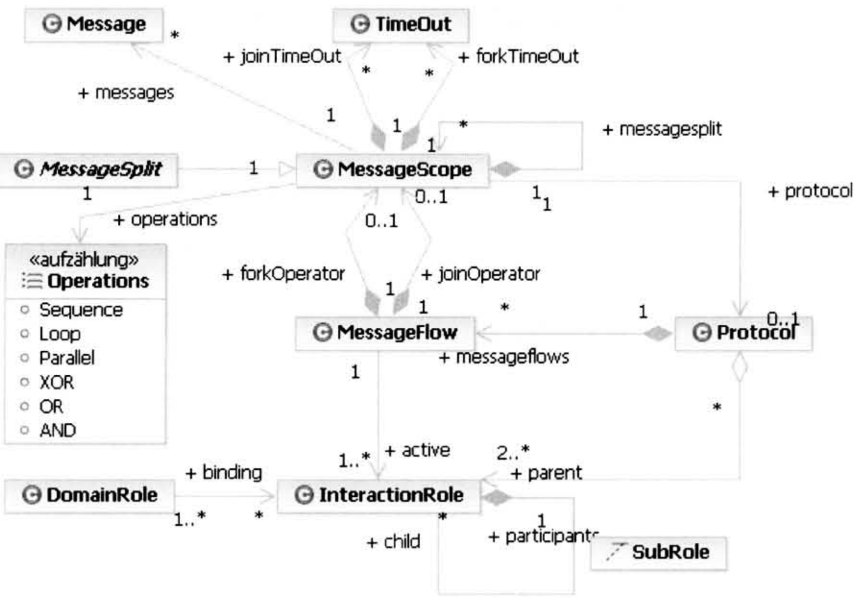

Fig. 10. The metamodel reflecting the interaction aspect of the PIM4Agents.

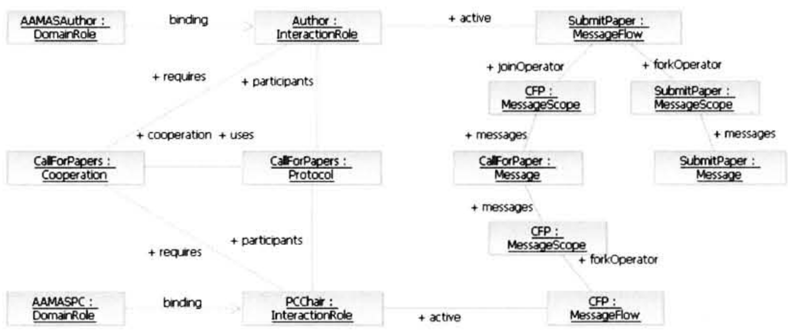

Fig. 11. Interaction model of the CMS.

Fig. 11 discusses the interaction model that covers the interaction between the authors and the PC in the submission phase. The CallForPapers Cooperation uses a CallForPapers Protocol and requires the InteractionRoles Author and PCChair. The PCChair is active in the CFP MessageFlow that refers via a CFP MessageScope to the CallForPaper Message, whereas the Author is active in the MessageFlow SubmitPaper that refers via a CFP MessageScope to a CallForPaper Message and via a SubmitPaper MessageScope to a SubmitPaper Message.

5 Metamodel for JACK

A vast number of frameworks and methodologies have been developed to foster the software-based development of BDI agent architectures [30] and MAS [31, 11, 32–34]. As mentioned in Section 3.4, JACK is a prominent example of a BDI implementation and is considered in our approach as platform-specific execution environment. The partial metamodel of JACK (JACKMM) is presented in the following section.

5.1 Team Metamodel

The team metamodel specifies and defines the structure of one or more entities that is formed to achieve a set of desired objectives. A subset of the metamodel for this aspect is presented in Fig. 17.

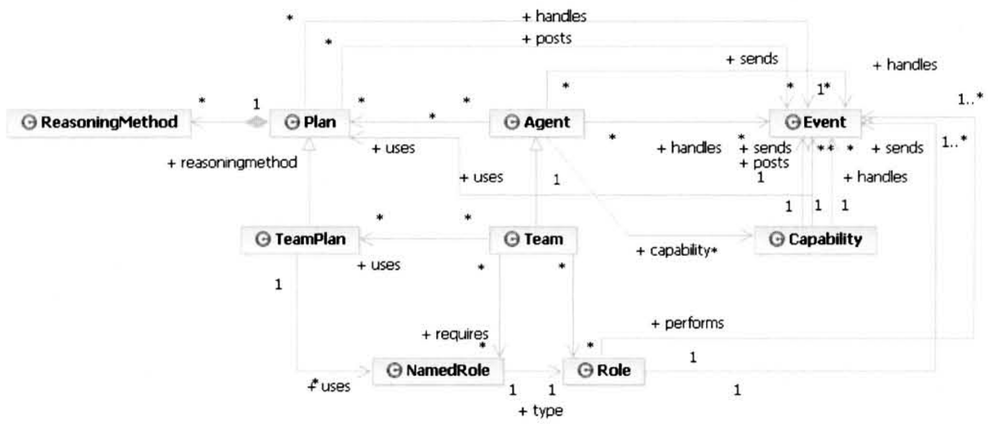

Fig. 12. The team metamodel reflecting the team aspect in the JACK framework.

An Agent is a component that can exhibit reasoning behaviour under both proactive (goal directed) and reactive (event-driven) stimuli. When an Agent is instantiated, it will wait until it is given a goal to achieve or experiences an Event that it must respond to. When such a goal or Event arises, it determines what course of action it will take. The Team concept is a specialization of Agent. It is a distinct reasoning entity which is characterized by the Roles it performs and the NamedRoles it requires others to perform. The formation of a given Team is achieved by attaching sub-teams capable of performing the NamedRoles required by the Team. A Plan models procedural descriptions of what an Agent does to handle a given Event. All the action that an Agent takes is prescribed and described by the Agent's Plans. A TeamPlan specifies the behaviour of a Team in reaction to a specific Event. As a specialization of Plan, a TeamPlan also defines a set of steps specifying how a particular task is achieved by particular NamedRoles. In order to coordinate the Team's behaviour, TeamPlan provides additional constructs like the team_achieve statement (for more details we refer to Section 5.2).Role definitions are a very important concept to define a Team as those specify which messages—which are rather Events— the role fillers are able to react to and which messages they are likely to send. An Event presets the type of stimuli a Team, Role, or TeamPlan reacts to or posts. JACK distinguishes between (i) *internal stimuli* that are events the Agent/Team sends to itself, (ii) *external stimuli* that are messages from other Agents, and (iii) *motivations* such as goals the Agent/Team may have. The details on the discussed concepts and their attributes are given in Table 1.

5.2 Process Metamodel

The process metamodel for JACKMM is presented in Fig. 13. It describes the process structure and the available language constructs for process definition. The concept Process illustrates the main part of the process aspect. It includes various occurrences of the type NodeBase which is an abstract class from which each particular node inherits. Furthermore, the Process comprehends a set of Flows that define the control flow between nodes. Each Flow has exactly one source node and one sink node. A complete list of all process-related concepts is given in Table 2.

JACK'S Team elements		
Concept	*Attributes*	*Explanation*
Agent	sends	Events are identified that the Agent sends externally to other Agents
	handles	Events that the agent will attempt to respond to if they arise by executing a Plan
	uses	Plan that the Agent can execute in reaction to an Event
Team	uses	TeamPlan the Team executes when handling an Event
	performs	Roles the Team performs itself to the outside
	require	NamedRoles the Team requires in order to solve the requested task
Plan	reasoningmethod	defines methods that an Agent may execute when it runs this Plan. Reasoning methods are different from normal Java methods in that they execute as finite state machines, and may succeed or fail, depending on whether the Agent can complete each statement that they contain. The top-level reasoning method is called *body*
	handles	Events that trigger the execution of the Plan
	posts	Events that are posted within a Plan
TeamPlan	uses	Roles that are needed by the TeamPlan to solve the assigned task
Role	handles	Events that are handled by a particular Role
	posts	Events that are posted by a particular Role
NamedRole	type	Role type that is referred by the NamedRole

Table 1. The Team viewpoint of JACK.

6 Metamodel for JADE

The JADE agent platform [5] is a very popular platform with the MAS community, therefore it was chosen as a relevant target platform to our MDD approach. This section presents a partial view of a metamodel for this platform. It is important to mention that, since JADE is implemented in Java, the Java language constructs (classes, interfaces, etc.) are also available, but not covered in detail in this paper.

6.1 Core View of JADEMM

The JADE metamodel (JADEMM) presents the concepts and structures available in the JADE API [35] and some minor extensions for mapping purposes. A reduced view of this core is shown in Fig. 14.

The Agent represents the class jade.core.Agent from the JADE API. The software agent performs various tasks, including message passing and the scheduling and execution of multiple concurrent activities. The Behaviour represents the codebase to all the actions that the agent can perform. Since it is the base of the Behavior model, it is abstract and its children are the ones that can actually be instantiated and executed. The Agent's knowledge is stored in an Ontology, which contains application specific concepts that Agents can use in their messages. It defines a vocabulary and relationships between the elements in this vocabulary. Correspondingly, the ConceptSchema is an expression that describes an entity with a complex structure that can be defined in terms of Slots. The ACLMessage is the base for Agent communication. It implements an ACL message compliant to the FIPA ACL Message Structure Specification [36] and is parameterized though *key:value* pairs. In order to support Agent Organizations two concepts are introduced as an extension to the JADE API. The Organisation represents a generic grouping of Agents, it enables a straightforward support of organizational structures from the PIM4Agents. The Organisation also provides the codebase for further specialized Organisations,

JACK's process elements		
Concepts	*Explanation*	*Attributes*
NodeBase	abstract class that provides the common attributes for node specializations	—
Process	main process class that contains all NodeBases and Flows	subprocesses: collection of NodeBases under this Process start: first NodeBase in the Process flows: Flows that are needed to connect the specific NodeBases in the Process
Flow	concept to link NodeBases	sink: refers to NodeBases that are the source of a Flow source: refers to NodeBases that are the sink of a Flow
ForkNode	abstract class that extends NodeBase for the support of alternative outputs	—
ParallelNode	represents the parallel statement node	parallelTasks: collection of tasks or processes that must be executed in parallel
PostNode	posts a message to the same Agent	event: Event to be posted
SendNode	sends a message to the another Agent	targetAgent: the name of the recipient agent for the sent Event
ReplyNode	replies to a message received by the Agent	originalMessage: message to which the reply responds
CodeNode	executes Java code within the Plan	code: Java code to be executed
DecisionNode	represents an if-else decision	condition: the condition to be evaluated in the decision
SubtaskNode	executes another Plan as subtask by posting an event	eventToPost: the Event to be fired
SubgraphNode	executes a reasoning method as subpart of the process	subgraphNameAndArgs: the name and arguments for invoking the reasoning method
TestNode	test a given condition, if the value of the expression is unknown to the Agent a subtask is fired by posting an Event	condition: the condition to be evaluated goalEventToPost: the Event to be posted if the value of the evaluation is unknown to the Agent
DetermineNode	iterates through all possible values that satisfy a logical condition until a goal subtask using these values succeeds	condition: the condition to be evaluated. goalEventToPost: the goal Event that the Agent executes for each set of values that satisfy the binding condition.
AchieveNode	asks the Agent to test a condition and if it is not true, to handle a goal Event	condition: the condition to be evaluated goalEventToPost: Event describing the goal that the Agent must try to achieve
InsistNode	similar to achieve, but ensures that the condition holds after the execution of the goal subtask	
MaintainNode	similar to SubtaskNode, but ensures that a condition is held during the execution of the subtask	condition: the condition to be held eventOrReasoningMethod: Event that fires the subtask or reasoning method

Table 2. The process elements of JACK

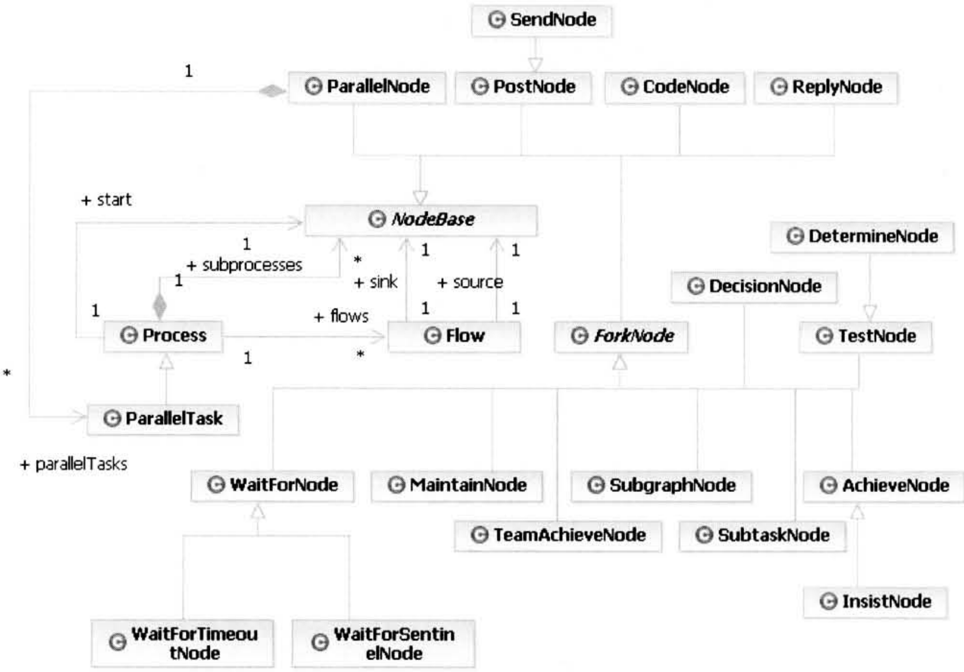

Fig. 13. The partial process metamodel reflecting the process aspect in the JACK framework.

such as holons for instance. The members of the Organisations are characterized by Roles, which describe identify the part they play within the Organization. For more details on the concepts please refer to Table 3 and [35].

JADEMM Core elements		
Concept	*Attributes*	*Explanation*
Agent	ontology	representation of the Agent's knowledge, necessary for message processing using templates
	behaviours	set of possible actions that the Agent can execute
	implements	set of Roles implemented by the Agent
Organisation	members	Agents that take part in the Organisation
	requires	Roles the Oraganisation needs to achieve its tasks
Role	sends	Messages that the Role may send
	receives	Messages that the Role may receive
Ontology	schemas	Schemas that the ontology contains
ACLMessage	performative	ACL performative that the message performs

Table 3. The core aspect of JADEMM

6.2 Behaviour View of JADEMM

The Behaviour, previously introduced in the core of JADEMM, represents any process or task that can be executed by the Agent. It is an abstract class, but it is the base for various specialized behaviour types. We mainly concentrate on two types of them: SimpleBehaviour and CompositeBehaviour. These two types are abstract and provide the base class for additional specializations, simple or composite behaviors correspondingly. A small extension was added to the hierarchy to represent the sending and reception of messages by the MessageReceiverBehaviour and MessageSenderBehaviour. A partial view of the Behaviour hierarchy is depicted in Fig. 15

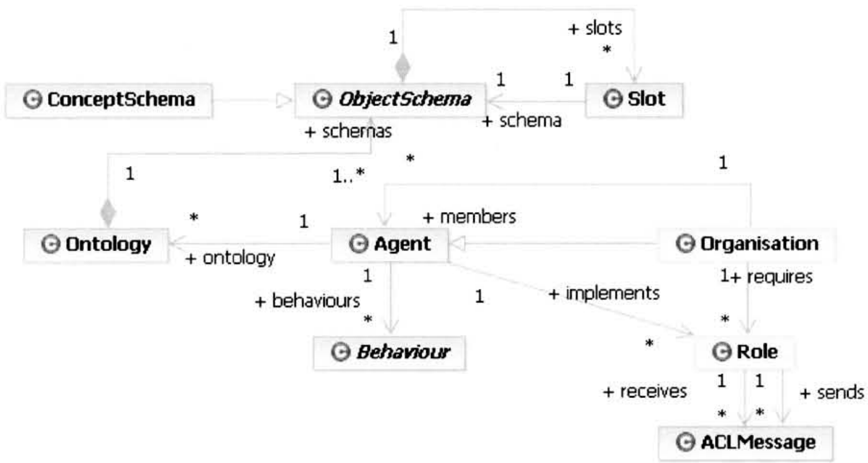

Fig. 14. Partial view of the core of the JADE metamodel

and a summarized description of the most relevant specializations in the behaviour hierarchy is presented in Table 4.

JADEMM Behaviour elements		
Parent Behaviour	*Behaviour Type*	*Explanation*
SimpleBehaviour	OneShotBehaviour	represents an action that is performed once only.
	CyclicBehaviour	represents an action that is performed indefinitely
CompositeBehaviour	ParallelBehaviour	executes its children in parallel fashion, and concludes when a predetermined number, all or any of its children are done.
	FSMBehaviour	is a serial behaviour that executes its children according to a FSM defined by the user. More specifically each child represents a state in the FSM.
	SequentialBehaviour	is a serial behaviour that executes its children in sequential order, and terminates when its last child has ended.

Table 4. The Behaviour Aspect of JADEMM

7 Vertical Transformations

Model transformations are one of the key mechanism within MDD. Using code generation templates, the model is transformed to executable code that may be optionally be merged with manually written code. One or more model-to-model transformation steps may precede the code generation. These model-to-model transformations can be distinguished between vertical (between PIM and PSM) and horizontal (between PIM and PIM) mappings. This section deals with vertical mappings, i.e., how to map PIM-related concepts (defined by the PIM4Agents metamodel) to PSM-related concepts of JACKMM and JADEMM.
The mapping rules we are discussing in the following are defined on the basis of the source and target metamodel, whereas the execution, i.e. the transformation of them is done on the source and target models. The mapping rules consist of (i) a head that defines which concepts from the source metamodel are mapped to which concepts of the target metamodel and (ii) a body that defines how the attribute's information of the target metamodel is derived.

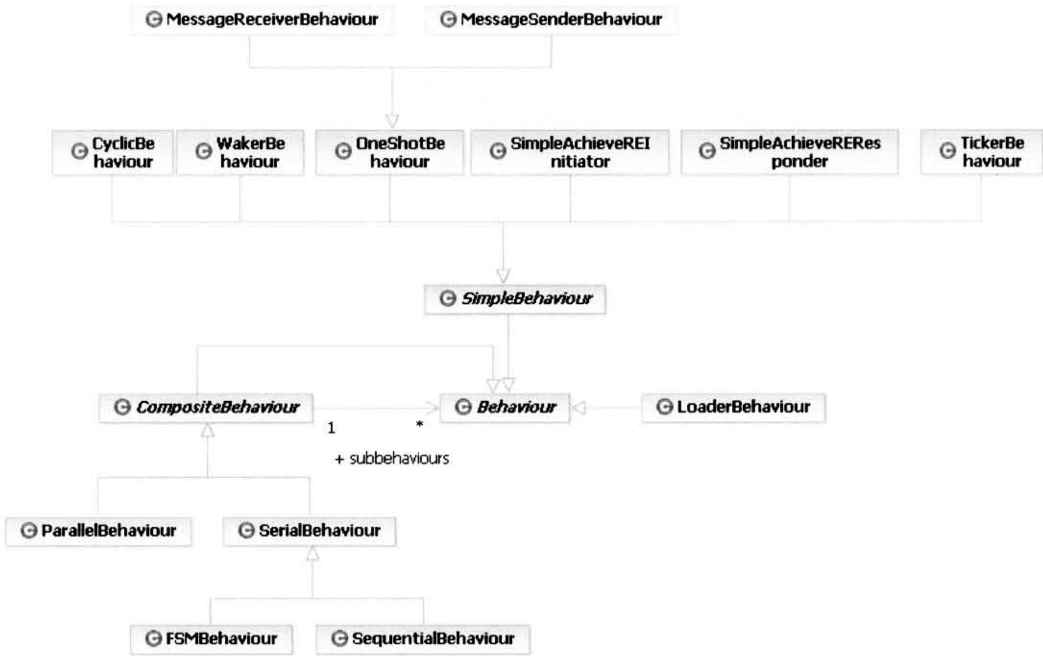

Fig. 15. Partial view of the behavior hierarchy of the JADE metamodel

7.1 From PIM4Agents to JACKMM

In this section we bring together the metamodels of the PIM4Agents (see Section 4) and JACK (see Section 5). Therefore, several basic mapping rules were defined that are listed in the remainder of this section. The first rule covers the mapping from the organization aspect (i.e. the concept Organization and its attributes) of the PIM4Agents to the team aspect of JACKMM. Therefore, we have defined the following mapping rule.

Model Mapping 1:

Head: $PIM4Agents.Agent : Organisation \rightarrow JACKMM.Team : Team$

Body: The Behaviour that is used by the Organisation is mapped to a set of TeamPlans the Team makes use of. The order in which Plans are executed is only mapped for these Plans in the PIM4Agents that do not react on an incoming Message. As the execution order in JACKMM is mainly predefined by the order in which Events are sent and handled by the TeamPlans. Events a Team sends or handles are extracted from the organizational Protocol. The Team performs and requires Roles that are specified by the Organization's provided DomainRoles and required InteractionRoles. The body function of this mapping rule is discussed by Table 5 in more detail.

The source and target concepts of Mapping Rule 1 nicely corresponds to each other as both (i) make use of a process that specifies how their members are coordinated and (ii) require and perform Roles, even if we distinguish between DomainRoles and InteractionRoles in the PIM4Agents. The only difference between both metamodels is the manner in which interactions are defined. In general, the interaction in the PIM4Agents is defined by a Protocol whereas JACKMM defines the interaction between entities in an event-driven manner without explicitly specifying a protocol. The mapping between the interaction aspect and the event-driven manner provided by JACKMM is one of the more difficult mappings that is discussed in more detail in Mapping Rule 4. The second transformation rule deals with the mapping from the agent aspects of the PIM4Agents to the team aspect in JACKMM.

$PIM4Agents.Agent : Organisation \rightarrow JACKMM.Team : Team$		
Target	*Source*	*MR*
Team.performs	DomainRoles that are performed by the Organization	7
Team.requires	DomainRoles that are performed by the Organization's members	6
Team.handles	collection of all Process' Messages that are received by the Interaction-Roles the Organization's DomainRoles are bound to	4
Team.sends	collection of all Process' Messages that are sent by the InteractionRoles the Organization's DomainRoles are bound to	4
Team.uses	collection of all Steps that are (i) contained in the Organization's Behaviour and of the type Plan	3
Team.capability	Capabilities that are used by the Organizations	5

Table 5. Mapping Rule 1 in detail.

Model Mapping 2:

Head: $PIM4Agents.Agent : Agent \rightarrow JACKMM.Team : Team$

Body: The Behaviour that is used by the Agent is mapped to a set of TeamPlans the Team makes us of. The Protocol's Messages the Agent participates are mapped to Events that are either handled or sent by the Team. Furthermore, the Team performs the Roles that are defined by the InteractionRoles the Agent's DomainRole is bound to in the PIM4Agents model. The details of the mapping body are discussed by Table 6.

$PIM4Agents.Agent : Agent \rightarrow JACKMM.Team : Team$		
Target	*Source*	*MR*
Team.performs	DomainRoles that are performed by the Agent	6
Team.requires	—	—
Team.handles	collection of all Process' Messages that are received by the Interaction-Roles the Agent's DomainRoles are bound to	4
Team.sends	collection of all Process' Messages that are sent by InteractionRoles the Agent's DomainRoles are bound to	4
Team.uses	collection of all Steps that are (i) contained in the Agent's Behaviour and (ii) of the type Plan	3
Team.capability	Capabilities that are used by the Agent	5

Table 6. Mapping Rule 2 and its details.

At first glance the concept Agent of JACKMM seems to be the best match, but since an Agent in the PIM4Agents references Roles, it is recommended to assign PIM4Agents.Agent:Agent to a Team in JACKMM as an Agent in the JACKMM does not refer to any Roles (see Fig. 17). The main difference between Mapping Rule 2 and Mapping Rule 1 is the fact that when mapping an Agent to a Team we instantiate an atomic Team which means that the Team does not require any NamedRole to which tasks could be assigned in the TeamPlan. When mapping an Organization, the Team requires a set of InteractionRoles that are performed by the organizational members, where a member could also be of the type Organization.
The third mapping rule covers the mapping between the behavioural aspect of the PIM4Agents and the process aspect of JACKMM.

Model Mapping 3:

Head: $PIM4Agents.Behaviour : Plan \rightarrow JACKMM.Team : TeamPlan$

Body: A TeamPlan uses a set of NamedRoles that are extracted from the InteractionRoles an Organization/Agent in the PIM4Agents requires. In fact, only a Cooperation (and Organization that inherits from the Cooperation) requires InteractionRoles. So that the set of InteractionRoles an Agent requires would be empty. However, an atomic Team should not require any NamedRole. The Conditions are mapped to the triggering conditions in a TeamPlan. Additionally, the specializations of a Scope in the PIM4Agents are nearly mapped in a one-to-one fashion to the corresponding concepts of the JACKMM Process. The details of the body are specified in Table 10.

$PIM4Agents.Behaviour : Behaviour \rightarrow JACKMM.Team : TeamPlan$		
Target	*Source*	*MR*
TeamPlan.uses	InteractionRoles that are required by the Organization/Agent	6
TeamPlan.sent	Messages that are sent within a Protocol, i.e. Messages that are referred by the Plan's SendMessage	4
TeamPlan.handles	Messages that are handled within a Protocol, i.e. Messages that are referred by the Plan's ReceiveMessage	4

Table 7. Mapping Rule 3 and its details.

Process mappings		
Source	*Target*	*Explanations*
Process	Plan	the first Step (start) inside a Behavior is not explicitly represented in the PIM4Agents. Instead, we are mapping the Step that has no ingoing Flow. The subprocesses and flows are represented by the Plan's flows and steps.
Flow	Flow	by connecting the NodeBases using Flows we can easily represent a Sequence in the PIM4Agents
ParallelNode	Parallel	depending on the execution type (XOR, AND), we set the condition of the ParalledNode to ANY or ALL
SendNode	SendMessage	the Event that is sent in the SendNode is used to instantiate the corresponding Message in the PIM4Agents
CodeNode	InternalTask	statements inside an InternalTask are transformed to CodeNode
DecisionNode	Decision	the Condition in the PIM4Agents is mapped to the condition in JACKMM

Table 8. Mapping between the PIM4Agents and JACKMM process parts.

A Behaviour in the PIM4Agents consists of several Steps that are linked via a Flow. A Plan—which is one frasible specialization of a Step—unions Scopes that define more complex control structures and atomic Tasks like sending a Messages. As a specialization of Step, all three concepts (i.e. Plan, Scope and Task) refer to a set of incoming and outgoing Flows. How to map the particular concepts of PIM4Agents is illustrated in Table 8. In principle, a mapping rule has to be defined for each of them. We have chosen a simplified form of presentation since those rules are nearly mapped in an one-to-one manner.
The fourth mapping rule defines how to map the interaction aspect of the PIM4Agents that describes how to specify the interaction in a protocol-driven manner to an event-driven manner as it is supported by JACKMM.

Model Mapping 4:

Head: $PIM4Agents.Interaction : Message \rightarrow JACKMM.Team : Event$

Body: Each Message that is either part of a Protocol or is referred by an atomic Task (i.e. SendMessage or ReceiveMessage) in a Plan is mapped to an Event in JACKMM. This is done independent of its type, i.e. whether the Message is sent/received in an asynchronous or synchronous manner.

As mentioned in Section 4.4, JACK distinguishes between several different types of Events. In the case of Mapping Rule 4 we mainly concentrate on MessageEvents. GoalEvents are not covered as the PIM4Agents core does not yet present any goal-oriented concepts.

Model Mapping 5:

Head: $PIM4Agents.Interaction : Capability \rightarrow JACKMM.Team : Capability$

Body: The Behavior that is used by the Capability in the PIM4Agents is mapped to the handled Capability's Plans in JACKMM. The Messages that are sent and received within the particular Behavior are mapped to Events that are sent and handled by the Capability in JACKMM.

$PIM4Agents.Interaction : Capability \rightarrow JACKMM.Team : Capability$		
Target	*Source*	*MR*
Capability.handles	Messages that are handled within the Plans that are grouped by the Capability in the PIM4Agents	4
Capability.sends	Messages that are sent within the Plans that are grouped by the Capability in the PIM4Agents	4
Capability.posts	—	
Capability.uses	Behaviour that is referred by the Capability in the PIM4Agents	3

Table 9. Mapping Rule 5 and its details.

The concept Capability is used by the Agent and Role in the PIM4Agents to group a particular type of Behaviour. The manner in which the Capability is used in JACK nicely corresponds to this. However, only the concepts Agent and Team refer to Capabilities, Roles do not have a pointer to Capabilities in JACKMM. To compensate this, we additionally have to introduce Capabilities for those Agents and Teams that perform the particular Role in the PIM4Agents.

Model Mapping 6:

Head: $PIM4Agents.Agent : InteractionRole \rightarrow JACKMM.Team : Role$

Body: The concept InteractionRole of the PIM4Agents is transformed to JACK-related Roles a Team *requires* or *performs.*

Model Mapping 7:

Head: $PIM4Agents.Agent : InteractionRole \rightarrow JACKMM.Team : NamedRole$

Body: For each InteractionRole that is specified within a Protocol a Role in JACKMM is instantiated. The NamedRole refers to the particular Role that is introduced by Mapping Rule 6.

PIM4Agents.Agent : InteractionRole → JACKMM.Team : Role		
Target	*Source*	*MR*
Role.handles	Messages that are handled by the InterationRoles the corresponding DomainRole is bound to	4
Role.posts	Messages that are sent by the InterationRoles the corresponding DomainRole is bound to	4

Table 10. Mapping Rule 6 and its details.

The PIM4Agents distinguishes between two different role types. The DomainRole focuses more on the Role an Agent/Organization is able to play within a certain domain. The InteractionRole focuses more on the Role an Agent/Organization is able to play within a Cooperation. Consequently, a DomainRole could play more than one InteractionRoles and an InteractionRole could be played by several Agents/Organizations at the same time. The DomainRoles that are bound to the particular InteractionRoles are used as role fillers, i.e. they perform the Role to which InteractionRole they are bound. In JACK, the Roles required by a Team are rather represented by role container objects, which include the Role objects as fillers.

Model Mapping 8:

Head: *PIM4Agents.Agent : Resource → JACKMM.Team : NamedData*

Body: Resources an Agent has access to in the PIM4Agents are mapped to NamedData an Agent or Team uses. The NamedData concepts refers to so-called external classes that are specified in e.g. Java.

7.2 Generated JACKMM models

In the previous section, we illustrated the basic mapping rules used to transform PIM4Agents models to JACK models. For the purpose of demonstration, we relate this model mapping to the PIM4Agents models that were discussed in Section 4 and explain how the generated JACK models look like.
Fig. 16 depicts the output model when applying the particular mapping rules on the PIM4Agents model illustrated by Fig. 3. In particular, applying Mapping Rule 3 generates a TeamPlan HandleCFP that is referred by the Capability AuthorCapability that is instantiated by applying Mapping Rule 5. Furthermore, Mapping Rule 2 generates three Team instances (AuthorAgent1, AuthorAgent2 and AuthorAgent3) that perform the same Role and make use of the same Capability AuthorCapability. Finally, Mapping Rule 6 generates the Role instance Author.

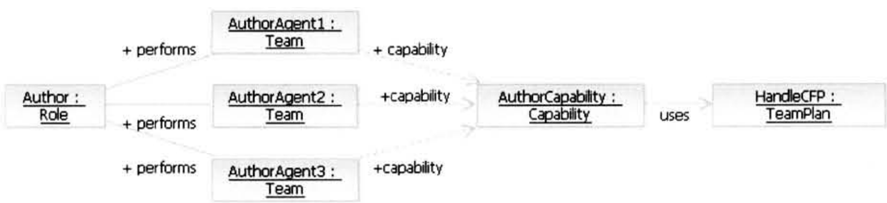

Fig. 16. The generated JACK model that bases on the agent model illustrated in Fig. 3.

Fig. 17 depicts the output model when applying the particular mapping rules on the PIM4Agents model illustrated by Fig. 5. In particular, applying Mapping Rule 1 generates an instance of an

Organization called PC. The body of this mapping refers to a set of Capabilities that are generated by applying Mapping Rule 5. Furthermore, the PC Team requires a set of NamedRoles (i.e. PCMember and PCChair) that are generated by applying Mapping Rule 7. These NamedRoles refer to the Roles PCMember and PCChair (Mapping Rule 6). Using Mapping Rule 2, we generate the Teams PCMemberAgent2, PCMemberAgent1 and PCChair that base on the agent types in the PIM4Agents CMS model. The Teams PCMemberAgent1 and PCMemberAgent1 perform the Role PCMember, whereas the PCChair performs the PCChair Role. The Team PCChair has a Capability PCChairCapability, the PCMemberAgent1 and PCMemberAgent2 have a Capability PCMemberCapability. Both Capabilities are instantiated by Mapping Rule 5. The PCMemberCapability uses a Behaviour Review, the PCChairCapability uses a Bahaviour ReceiveSubmission (Mapping Rule 3).

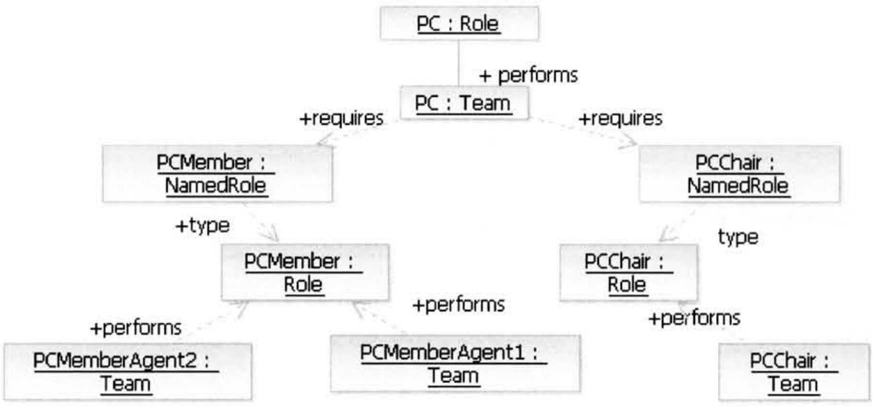

Fig. 17. The generated JACK model that bases on the organization model illustrated in Fig. 5.

Fig. 18 depicts the output model when applying the particular mapping rules on the PIM4Agents model illustrated in Fig. 9. In particular Mapping Rule 3 is mainly responsible for the newly instantiated NodeBases in Fig. 18. The first Step that is neither a SendMessage (i.e. the TeamPlan handles this Event) nor a Sequence (i.e. this Step is implicitly illustrated by the Flow concept in JACKMM) is presented as start attribute (i.e. DecisionNode WritePaper), the others are subprocesses. This DecisionNode is linked to the CodeNodes Relax and WritePaper via the Flow FlowInstance. Like the Steps, the Flows are also included BodyReasoningMethod. Exemplarily, this is shown by the 'flows' associaitions between the HandleCFPPlanBodyReasoning and the FlowInstance.

7.3 From PIM4Agents to JADEMM

This section introduce the mapping from the PIM4Agents concepts (Section 4) to the JADEMM concepts presented in Section 6 through various mapping rules. The list presented does not comprehend all the necessary model mapping, but only the most relevant for a clear understanding of how they are applied for the presented model mappings.

Model Mapping 9:

Head: $PIM4Agents.Agent : Agent \rightarrow JADEMM : Agent$

Body: Every Agent in the PIM4Agents is mapped to a *JADEMM:Agent.* The details of this mapping rule are summarized by Table 11.

The $PIM4Agents.Agent : Agent \rightarrow JADEMM : Agent$ Mapping is fairly straight forward, given that the concepts correspond to one another in the use of behaviours, to carry actions;

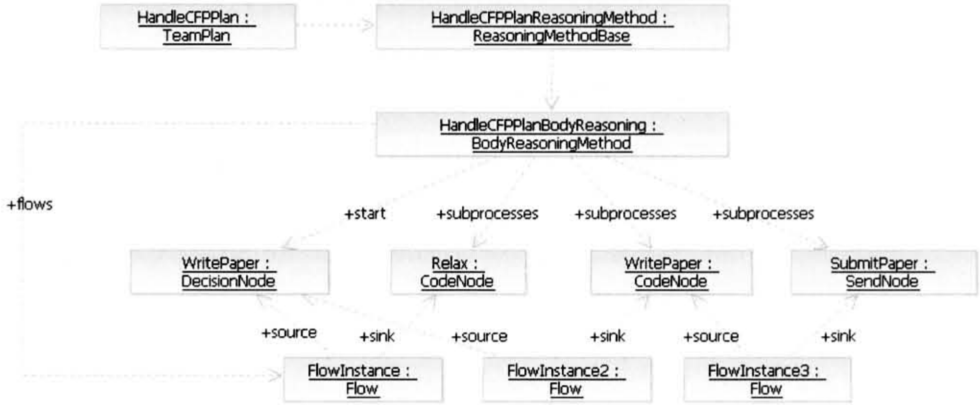

Fig. 18. The generated JACK model that bases on the behaviour model illustrated in Fig. 9.

$PIM4Agents.Agent : Agent \rightarrow JADEMM : Agent$		
Target	*Source*	*MR*
Agent.implements	collection of DomainRoles that are performed by the Agent	12
Agent.behaviours	collection of Behaviours that determine what the Agent can do, obtained from the Behaviors the Agent has and the Capabilities the Agent use	14,13
Agent.organization	collection of Organizations that the Agent is a member of	10

Table 11. Model Mapping 9 in detail.

Roles, to represent responsibilities or compromises; and Organizations, to collaborate with other Agents.

Model Mapping 10:

Head: $PIM4Agents.Agent : Organisation/Cooperation \rightarrow JADEMM : Organisation$

Body: *JADEMM.Organisation*, an extension to the JADE API, allows to transform *PIM4Agents.Agent:Organisation/Cooperation* in the straightforward fashion that is presented in Table 12.

$PIM4Agents.Agent : Organisation/Cooperation \rightarrow JADEMM : Organisation$		
Target	*Source*	*MR*
Organisation/Cooperation.-members	collection of Agents or Organizations that form this Organization, obtained from the members of this particular Cooperation	9
Organisation/Cooperation.-requires	collection of DomainRoles that the organization needs for its operation, obtained from all DomainRoles that are bound to the particular InteractionRole	12

Table 12. Model Mapping 10 in detail.

The concepts of an Organization or Cooperation in the PIM4Agents are mapped directly to JADEMM:Organisation, since the concept in JADEMM is a custom made extension to the JADE API, therefore its properties are mainly mapped in a one-to-one fashion. Although the transformation itself is not complicated, ensuring that the 'implementation/runtime version' of the Organisation performs the expected tasks requires some care at the technical programming level. Currently, it is a quite simple implementation and will evolve as more scenarios impose additional technical requirements on it.

Model Mapping 11:

Head: $PIM4Agents.Interaction : Protocol \rightarrow JADEMM : FSMBehaviours$

Body: The *PIM4Agents.Interaction:Protocol* is decomposed into n *JADEMM.FSMBehaviour* types–one for each InteractionRole in the Protocol—whose execution order is determined by the *PIM4Agents.Interaction:MessageFlow* for corresponding Role. The details for this mapping are shown in Table 15.

$PIM4Agents.Interaction : Protocol \rightarrow JADEMM : FSMBehaviours$		
Target	*Source*	*MR*
FSMBehaviour.name	the name of the FSMBehaviours is defined by the concatenation of the Protocol's name, the InteractionRole's name and the string 'Behaviour'	—
FSMBehaviour.children	the children behaviours are set by grouping the Protocol's Behaviours according to Messages that are sent and reacted to with respect to the Role's MessageFlow.	12
FSMBehaviour.transitions	the transitions from one child to the next are set by linking the forkOperator and joinOperator of a MessageFlow for the corresponding Role.	—

Table 13. Model Mapping 11 in detail.

As presented, Model Mapping 11 is a much more complex mapping than the ones presented so far. It basically does a collapse of the 'MessageFlow graph' and links the Scopes that correspond to each MessageFlow in the PIM4Agents into a set of FSMBehaviours in the JadeMM, whose transitions depends on the graph's links. Which Scopes should go into the each of the FSMBehaviours depends on the InteractionRole in the PIM4Agents to which they belong.

Model Mapping 12:

Head: $PIM4Agents.Agent : DomainRole \rightarrow JADEMM : Role$

Body: Every Role performed by an Agent is represented by an extension to the Jade API which contains the Role associated information, in particular the Messages that the Role sends and receives. A short explanation on the extraction of these message list is shown in Table 14.

The Role transformation (Model Mapping 12) also performs a collapse of the 'MessageFlow graph', but in this case, it groups the incoming and outgoing Messages found in the graph with respect to the InteractionRole. Additionally, the InteractionRoles are unified with the DomainRoles through the DomainRole.binding property, therefore there is only one Role concept in JADEMM which models the Interaction and DomainRole concepts.

Model Mapping 13:

Head: $PIM4Agents.Behaviour : Behaviour \rightarrow JADEMM : SequentialBehaviour$

Body: *JADEMM.Behaviour* is an abstract class, so the target for the transformation of the Behaviour is actually the SequentialBehaviour in JADEMM.

PIM4Agents.Agent : DomainRole → JADEMM : Role		
Target	*Source*	*MR*
Role.sends	in order to obtain the messages to be sent by the *JADEMM:Role*, we navigate the MessageFlows of the Protocol of the associated InteractionRole (PIM4Agents.Agent:DomainRole.binding). MessageFlows that possess a forkOperator value other than null, define a message to be sent by the Role.	—
Role.receives	in a similar fashion, in order to obtain the messages to be received by the Role in JADEMM, we navigate the MessageFlows of the Protocol of the associated InteractionRole (PIM4Agents.Agent:DomainRole.binding). MessageFlows that possess a joinOperator value other than null, define a message to be sent by the Role.	—

Table 14. Model Mapping 12 in detail.

PIM4Agents.Behaviour : Behaviour → JADEMM : SequentialBehaviour		
Target	*Source*	*MR*
SequentialBehaviour.-children	the children behaviours are obtained from the Plans in the PIM4Agents, and the order of the children is determined by the ControlFlow defined in the particular Plan	15

Table 15. Model Mapping 13 in detail.

Model Mapping 13 represents the general rule for mapping behaviours. In practice there are several mapping rules for each particular specialization of Behaviour presented in the PIM4Agents.

Model Mapping 14:

Head: *PIM4Agents.Agent : Capability → JADEMM : Behaviour*

Body: For every *PIM4Agents.Behaviour:Behaviour* contained in the *PIM4Agents.Agent-:Capability* referenced, a *JADEMM.Behaviour* will be added to the available behaviours of the Agent.

Model Mapping 15:

Head: *PIM4Agents.Behaviour : Scope → JADEMM : CompositeBehaviour*

Body: *PIM4Agents.Behaviour:Scope* is not transformed directly, for it is an abstract concept, nevertheless its subclasses are mapped to different CompositeBehaviours in the JADEMM in a somewhat straightforward manner. The general details of this mapping are shown in Table 16.

In similar fashion to Mapping Rule 13, Mapping Rule 16 represents a series of specific rules for transforming particular specialized types of Scopes. For example a Sequence in the PIM4Agents is transformed in SequentialBehaviour or ParallelBehaviour in JADEMM.

Model Mapping 16:

Head: *PIM4Agents.Behaviour : Task → JADEMM : OneShotBehaviour*

Body: The subclasses of the Task concept are mapped into OneShotBehaviours in JADEMM with different Java calls in their body corresponding to the task required. In the concrete cases of the tasks ReceiveMessage and SendMessage, they will be mapped to a MessageReceiverBehaviour and a MessageSenderBehaviour correspondingly.

$PIM4Agents.Behaviour : Scope \rightarrow JADEMM : CompositeBehaviour$		
Target	*Source*	*MR*
CompositeBehaviour.-children	if a Scope in the behavioural aspect of the PIM4Agents has sub-scopes, the children behaviours are generated according to these sub-scopes. The order of the children is determined by the outgoing and ingoing Flow of this Scope. If there are no sub-scopes are available, the children behaviours are generated based on the Scope's Steps	15, 16

Table 16. Model Mapping 13 in detail.

Model Mapping 17:

Head: $PIM4Agents.Agent : Message \rightarrow JADEMM : ACLMessage$

Body: PIM4Agents.Agent:Message is transformed to a ACLMessage in JADEMM with an INFORM performative as default. Depending on specific message types, other performatives may be used.

Model Mapping 18:

Head: $PIM4Agents.Agent : Resource \rightarrow JadeMM : ConceptSchema$

Body: PIM4Agents.Agent:Resources are transformed into ConceptSchema with the corresponding slots depending on the resource.

7.4 Generated JADEMM models

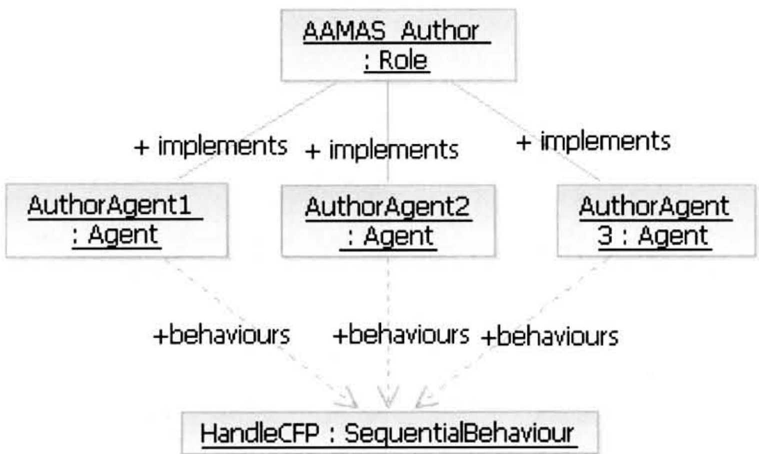

Fig. 19. The Agent View of the Example in JADEMM

Fig. 19 presents the result of transforming Fig. 3. We can see how Model Mapping 9 was applied to the PIM4Agents.Agent:Agents to obtain a JADEMM:Agents. We see the Capabilities

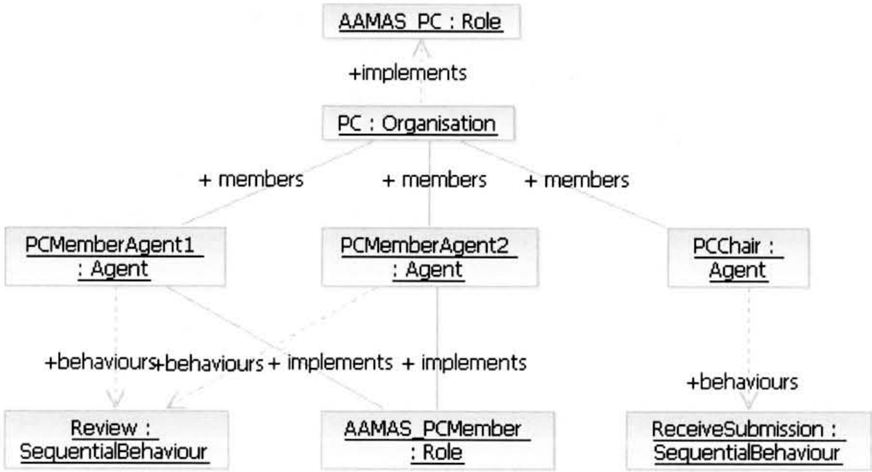

Fig. 20. The Organization View of the Example in JADEMM

disappear, but their behaviours are added to the corresponding Agents (Model Mapping 14). Additionally, we can see how Model Mapping 12 was applied to the *AAMAS_Author* Role. The transformed organization view from the example is presented in Fig. 20. Again, we see the DomainRoles—*AAMASPCMember*, *AAMASPCChair*, and *AAMASPC*—transformed in JADEMM:Roles through Model Mapping 12. Model Mapping 10 is then applied to *PC* to obtain a JADEMM:Organisation. Once again the behaviours linked to the Capability in PIM4Agents are linked directly to the corresponding Agents through Model Mapping 14. Additionally, *Review* and *ReceiveSubmission* are converted to JADEMM: SequentialBehaviours by Model Mapping 13.

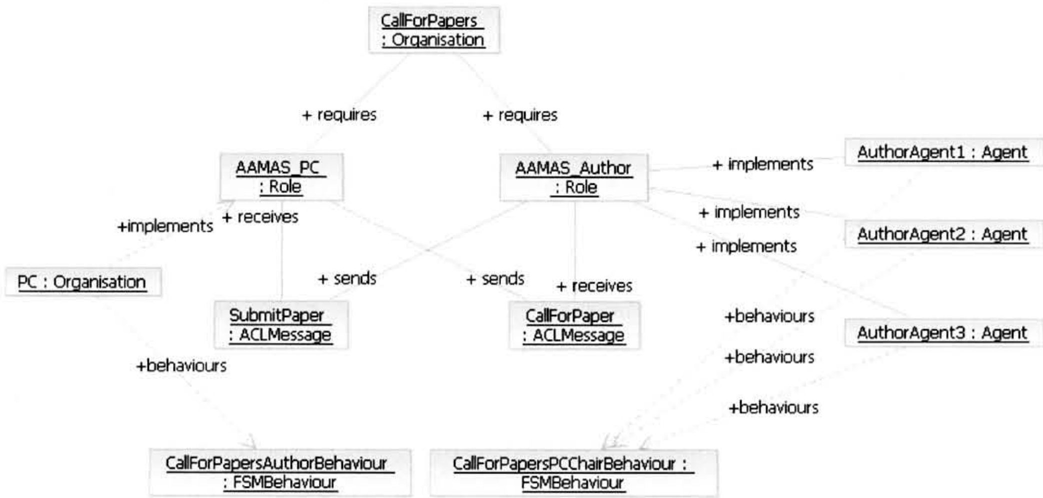

Fig. 21. The Interaction View of the Example in JADEMM

The interaction for the 'Call For Papers' process in JADEMM is depicted in Fig. 21. Once more, Agents are transformed by Model Mapping 9, DomainRoles by Model Mapping 12, and the Organisation by Model Mapping 10. The most relevant transformation in this view is the one of the Protocol *CallForPapers*. By the application of Model Mapping 11, the InteractionRoles are collapsed to their corresponding DomainRoles and the MessageFlow structure determines the contents of the output behaviours: *CallForPapersInitiatorBehaviour* and *CallForPapersResponderBehaviour*. These behaviours are liked to the corresponding role filler Agents/Organisations.

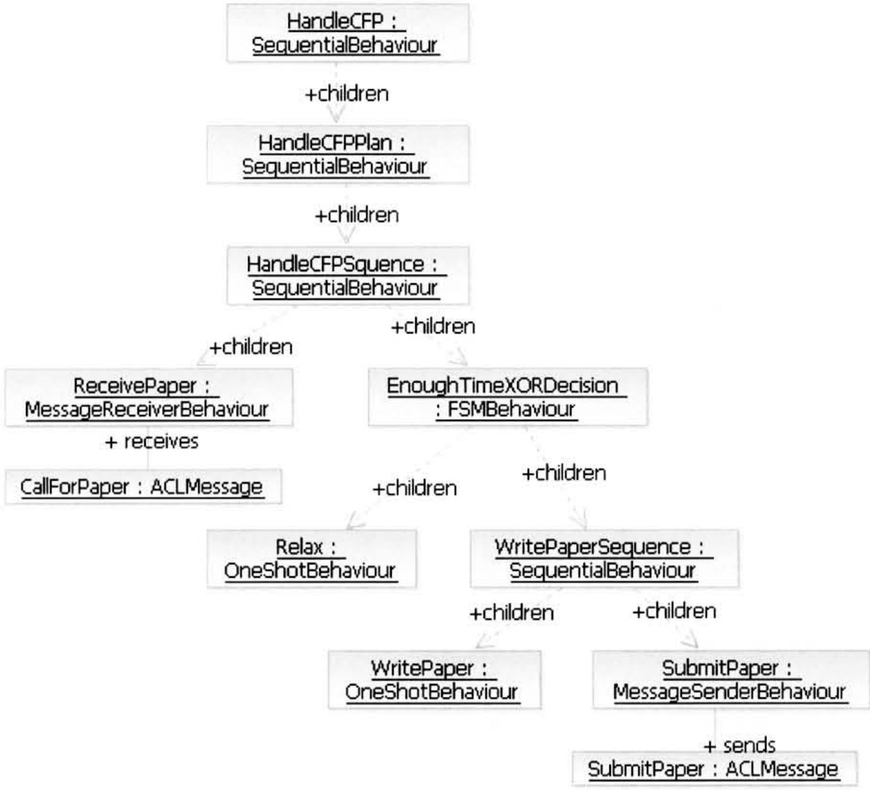

Fig. 22. The HandleCFP Behaviour in JadeMM

The HandleCFP Behaviour is presented in is JADEMM form by Fig. 22. By the application of Model Mappings 13, 15, 16, the PIM4Agents model presented in Fig. 9 is transformed to a JADEMM model. HandleCFP, HandleCFPPlan, HandleCFPSequence and WritePaperSequence are converted to SequentialBehaviours (Model Mappings 13 and 15). The XORDecision is converted to a FSMBehaviour also by Model Mapping 15. Finally, all Tasks—*ReceiveCFP*, *Relax*, *WritePaper* and *SubmitPaper*— are converted by Model Mapping 16.

8 Platform-Independent Model for Service-Oriented Architectures

Our proposed MDD approach allows to model agent systems using an abstract language that is defined by the PIM4Agents metamodel that can finally be executed by JACK or JADE using the model mappings we have defined in Section 7. This is one important step toward a domain specific language for agent systems. However, to integrate more application-oriented models into our approach is one further issue to make agent system more attractive for industry to adapt. With respect to this issue, we explored the possibility of integrating service-oriented architectures (SOA) into our MDD framework. Peer-to-Peer systems or grid systems are further attractive possibilities how to model modern information systems. In this paper, we base our approach on a metamodel for SOA [37] (called PIM4SOA) which has been developed by IBM, the European Software Institute (ESI) and SINTEF. The PIM4SOA covers four important aspects: service, process, information and quality of service.

Information: In the context of virtual enterprises information represents one of the most important elements that need to be described. In fact the other aspects manage or are based on information elements.

Service: Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity. In general, SOAs are formed by components provided by a system or a set of systems to achieve a shared goal.

Process: Processes describe a set of interactions among services in terms of messages exchange.

QoS: A suitable feature is the description and the modelling of non-functional aspects related with the services described.

8.1 Service Metamodel

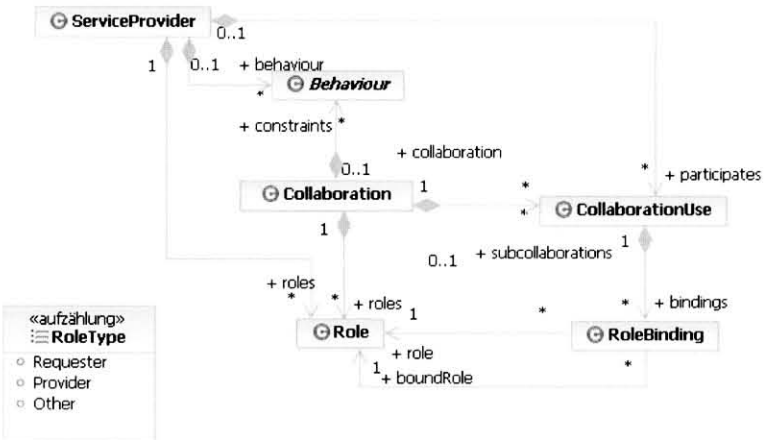

Fig. 23. The service metamodel of the PIM4SOA.

This section describes the elements in the service-oriented metamodel that has the objective of describing service architectures. These architectures represent the functionalities provided by a system or a set of systems to achieve a shared goal. These functionalities could be represented as a service or as a set of services. In this work we emphasize the concept of collaborations to address the different levels of service description. In this section we sketch out the main components of the service oriented metamodel. The service aspect of the PIM4SOA presents services modelled as collaborations that specify a pattern of interaction between the participating roles. A subset of the metamodel for this aspect is presented in Fig. 23.

A Collaboration represents a pattern of interaction between participating Roles. A binary Collaboration specifies a service. A Collaboration definition contains a set of Roles (provider, requester) and a set of CollaborationUses. Eventually it could be related with non-functional aspects. A Collaboration is related with a registry where endpoints are specified.

A CollaborationUse represents the usage of Collaboration. In other words, a CollaborationUse is the model element to represent a usage of a service. The CollaborationUse contains a reference to the endpoint pointing out the address. The concept RoleBinding relates a role with a usage of a service. When we specify a CollaborationUse we need to identify which are the Roles involved This relationship is made between two Roles: one inside the CollaborationUse and other inside a Collaboration definition.

A Behaviour is an abstract class for the specification of messages sequence within a service. This element represents a super class connecting a service aspect with process aspect. A ServiceProvider specify an entity describing and specifying in its turn services, roles and constraints. ServiceProvider represents a service specification containing the specification of other services. Non functional aspects could also be added to specify quality aspects. A Message defines a chunk of information sent from one Role to other Role in a Collaboration. A Message is owned by a specific Role.

8.2 Process Metamodel

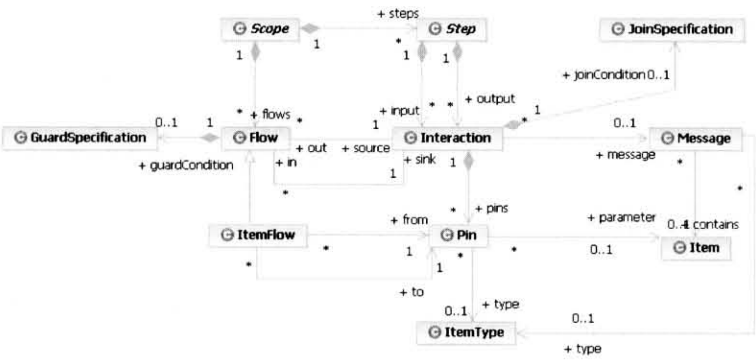

Fig. 24. The process flow of the PIM4SOA.

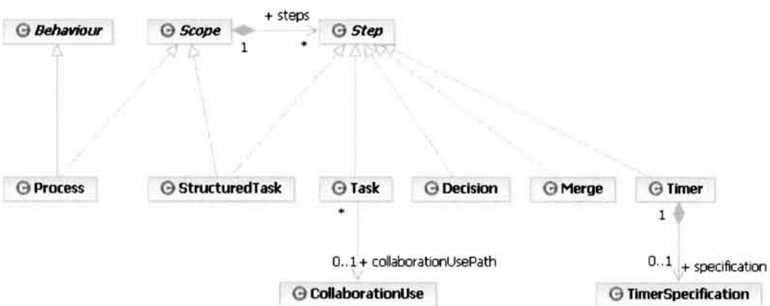

Fig. 25. The process elements of the PIM4SOA.

The process elements of the PIM4SOA metamodel are shown in Fig. 25. The process aspect is closely linked to the Service aspect, the primary link being the abstract class Scope above, which can be instantiated as a Process belonging to a ServiceProvider from that aspect.
The Process contains a set of Steps (generally Tasks), representing actions carried out by the Process. A Process consists of StructuredTasks (sub-processes), Steps (atomic tasks and actions, at the PIM level), and Interactions/Flows linking the Tasks together. These essentially fall into two categories, interactions with other ServiceProviders, or specialized actions requiring implementation beyond the scope of this model. For example, manual tasks to be processed by humans, or extensive computation requiring platform specific code.
The Process also contains a set of Flows between these actions, which may be specialized (ItemFlow) to indicate the transfer of specific data. This allows flexibility in that a business modeler may choose to start by showing only control flow, and later refine the model to include information. This links in to the Item/ItemType parts of the information aspect. Flows may diverge or reconverge using Guard and Join specifications.
The concept of a Scope is an abstract container for individual behavioural steps. This is subclassed only by Process and StructuredTask (Process is the top level behavioural object, StructuredTask may be used to group related Steps in a subroutine like manner.) A Step is a single

node in a Process, such as making a decision or calling an external service. The specialization of Step is Task. A Process implements a behaviour for a ServiceProvider, as a set of Tasks and Decisions (Steps) linked by control flows (Flows), optionally including detail on the exchanged messages / items.
A Task represents the low level building blocks of a process–these might be for example calls to another service (which can be transformed largely automatically to an implementation platform, with reference to the relevant Collaborations) or might require manual intervention–either in the form of hand coded functions, or human interaction with the process. An Interaction defines an interface for input or output flows on a Step. An Interaction can be considered as a set of Pins, though it is not compulsory to refine the model to this level (depending on aims of the model). If the Step is viewed as a service, this is similar to the declaration of a method/function in the interface (specifying a set of parameters or a return value).

Service aspect		
Concept	*Attributes*	*Explanation*
Collaboration	Subcollaborations	represent the usage of other Collaborations
	Constraints	constrain a Collaboration by the specification of a Process
	Roles	involved within the Collaboration
	Nfa	this element sets up a link to quality of service model definition
	Endpoint	is specified at design time
	RegistryItem	specify the registry item associated with the Collaboration
CollaborationUse	Provides	specify the provided item
	Messages	specify the Messages related with this Role
	RoleType	specifies the type of the Role. Basically a Role can be a requester or a provider. If it is not none of them we can specify it as 'other' and in the property Other we specify the name
	Other	used for the special case where the role is neither a requester nor a provider
RoleBinding	Role	represents a link to specific role within the collaboration definition of the current collaboration use
	BoundRole	represents a link to specific role within the current collaboration
ServiceProvider	Behaviour	represents the process
	Participates	contains a set of the collaboration uses
	Roles	defines the roles involved at this level
	Nfa	establishes the link to the quality of service model
	QosCategory	defines the category in terms of quality of service
	Type	refers to the type of provider: Abstract or Executable
Message	Contains	defines a set of items related with the Message
	Type	defines the type of the items related with the Message
	Mode	differentiates Messages between regular (normal) or fault (exceptions)

Table 17. The service aspects of the PIM4SOA.

9 Horizontal Transformations - From PIM4SOA to PIM4Agents

We already showed how to map the PIM4Agents metamodel to the JACK and JADE metamodels. We called these vertical transformations as the particular metamodels are situated on different abstraction levels. In this section, we discuss horizontal mappings between the PIM4SOA to the PIM4Agents—that are both considered as platform-indepedent—to allow that SOA can be deployed by agent systems. SOA and its corresponding metamodel (the PIM4SOA) describes IT system in a very abstract manner and thus provide a nice opportunity to illustrate how agent systems can be used in these kinds of environments in a model-driven development. By comparing the PIM4SOA and PIM4Agents metamodels, we derive the following basic mapping rules:

Model Mapping 19:

Head: $PIM4SOA.Service : Collaboration \rightarrow PIM4Agents.Agent : Organisation$

Body: For each Collaboration's Behaviour we generate an organisational Behaviour. Additionally, each ServiceProvider that participates in one of the CollaborationUses defines the organisational members. The Collaboration's Roles build the InteractionRoles. The Organisation requires a set of Protocols that are derived by extracting the message exchange in the Collaboration's or ServiceProviders's Behaviour.

$PIM4SOA.Service : Collaboration \rightarrow PIM4Agents.Agent : Organisation$		
Target	*Source*	*MR*
Organisation.requires	Roles that are referred by the Collaboration	24
Organisation.perfoms	—	
Organisation.behaviour	Behaviour that constraints the Collaboration in the PIM4SOA	21
Organisation.members	ServiceProviders that participates in a CollaborationUse that refers to this Collaboration	20

Table 18. The body of Mapping Rule 19 in details.

As Mapping Rule 19 nicely illustrates the concept of a Collaboration in the PIM4SOA corresponds to the concept of an Organization in the PIM4Agents as both refer to roles, processes that define their Behaviour and entities (i.e. ServiceProvider or Agents) that interact within those. However, the Collaboration does not perform any Role, so we do not instantiated any DomainRole that is performed by the Organization. However, the concept Organization seems to be the best match. Alternatively, we could use the concept of a Collaboration as it does not perform any DomainRole. However, Collaborations in the PIM4Agents do not refer to any Behaviour which might be necessary to map the Collaboration's Behaviour.

Model Mapping 20:

Head: $PIM4SOA.Service : ServiceProvider \rightarrow PIM4Agents.Agent : Agent$

Body: For each of the ServiceProvider's Roles we generate an Agent's *performed* DomainRole. The Behaviour is derived by extracting the ServiceProvider's Behaviour. The Agent's memberships are derived by extracting all Cooperations the particular ServiceProvider participates in.

Again, the concepts of the ServiceProvider can nicely be mapped to the corresponding concept of Agent in the PIM4Agents as the ServiceProvider performs a set of Roles, acts in accordance to some Behaviours and interacts with other ServiceProviders within a Collaboration.

Model Mapping 21:

Head: $PIM4SOA : Process \rightarrow PIM4Agents : Behaviour$

PIM4SOA.Service : ServiceProvider → PIM4Agents.Agent : Agent		
Target	*Source*	*MR*
Agent.perfoms	collection of Roles a ServiceProvider performs	23
Agent.behaviour	collection of the Behaviour that constraints the ServiceProvider	21
Agent.membership	collection of the CollaborationUses in which the ServiceProvider participates and the Collaborations they refer	19
Agent.has	—	

Table 19. The body of Mapping Rule 20 in details.

Body: The Process of the PIM4SOA is split into several Behaviours of the PIM4Agents. For each Task in the PIM4SOA that refers to a Message in an outgoing Interaction a new Behaviour is instantiated in the PIM4Agents. All Tasks that are connected via the outgoing Flow—directly of indirectly (i.e. via a Task that does not send a Message) are transformed to Plans in the PIM4Agents.

PIM4SOA : Process → PIM4Agents : Behaviour		
Target	*Source*	*MR*
Behaviour.steps	Steps that are contained in the PIM4SOA.Process	—
Behaviour.flows	Flows that are contained in the PIM4SOA.Process	—

Table 20. The body of Mapping Rule 21 in details.

The mappings between both process aspects is mainly straightforward as the PIM4Agents behaviour metamodels is more expressive.

Model Mapping 22:

Head: *PIM4SOA.Service : Message → PIM4Agents : Interaction : Message*

Body: The Messages specified inside the CollaborationUses and sent by the corresponding ServiceProvider's Roles are mapped to the Messages defining the Protocol.

Mapping Rule 22 is a straightforward mapping as the Message concepts of the PIM4Agents is kept in its core rather simple without referring to communicative acts (e.g. accept-proposal, refuse, etc.) or message parameters (e.g. content, language, etc.). These specializations could either be verbalized in further extensions that cover the compliance with FIPA or within the vertical mappings for those agent-oriented platforms that deals with FIPA-compliant concepts (for instance JADE).

Model Mapping 23:

Head: *PIM4SOA.Service : Role → PIM4Agents.Agent : DomainRole*

Body: Roles that are performed by ServiceProviders are mapped to DomainRoles. In each Collaboration the ServiceProvider participates, its Roles are bound to Collaboration's Roles. In Mapping Rule 24 these Collboration's Roles are mapped to InteractionRoles. The DomainRoles that are created by this Mapping Rule are bound to the particular InteractionRoles.

Model Mapping 24:

Head: $PIM4SOA.Service : Role \rightarrow PIM4Agents.Agent : InteractionRole$

Body: Roles to which ServiceProviders are bound to within a Collaboration are mapped to InteractionRoles.

A Collaboration refers to a set of CollaborationUses where each of them again refers to a Collaboration. In fact, the CollaborationUse links both Collaborations by binding the parent Collaboration's Role to the children Collaboration's Roles. Due to this recursion in modelling Collaborations, we do not translate each Role in a Collaboration to an InteractionRole in the PIM4Agents. In fact, for Roles that are bound to each other we introduce one InteractionRole.

Model Mapping 25:

Head: $PIM4SOA.Service : Collaboration \rightarrow PIM4Agents.Interaction : Protocol$

Body: A Protocol describes the message sequencing that is built by combining messages that are sent in the collaboration's collaboration uses. More precisely, the collaboration's Role types—requester and provider—are mapped to the Protocol's InteractionRoles, the Messages defined in the CollaborationUses are transformed to Protocol's Messages. Table 21 provides more details with respect to Mapping Rule 25.

$PIM4SOA.Service : Collaboration \rightarrow PIM4Agents : Interaction : Protocol$		
Target	*Source*	*MR*
Protocol.messageflows	Messages and how these are sent between the Roles within a Collaboration are extracted from the Collaboration's Behaviour and mapped to the Messages that are referred by the MessageScope and the Operations that defines in which manner those are sent.	22
Protocol.participants	Roles that are used by the Collaboration are selected to define the InteractionRoles that participates in the Protocol	24

Table 21. The body of Mapping Rule 25 in details.

Mapping Rule 25 is one of the more complex transformations, as the PIM4Agents does not provide any protocol-like viewpoint to define the ServiceProvider's interaction. However, this does not mean that an interaction cannot be described from a centralized viewpoint. A Collaboration's Behaviour could for instance be used to define the choreography's viewpoint. However, the information needed to initiate the PIM4Agent's Protocol needs to be extracted from various concepts.

Model Mapping 26:

Head: $PIM4SOA.Information : Document \rightarrow PIM4Agents.Agent : Resource$

Body: The information that is sent in Messages is defined by so-called Entities in the information metamodel. These Entities are part of Documents that are mapped to Resources in the PIM4Agents an Agent could have access to.

Documents mainly define how a service might look like in the PIM4SOA. At least they specify the service structure by defining Objects and their Attributes that then serve as input parameters to invoke particular services. This information is used to generate Resources an Agent has access to in the PIM4Agents. The set of accessible Resources are part of the Environment.
This section illustrated how to integrate domain-specific applications into the PIM4Agents using a MDD approach. We have discussed that a model mapping between the PIM4SOA

and PIM4Agents is realizable as the PIM4Agents is more expressive with respect to defining interactions and behaviour. Thus, PIM4SOA models can be transformed to PIM4Agents models that can be executed by JACK or JADE by applying the vertical mappings discussed in Section 7.

10 Technical Realization

Now that the transformations have been described, the details of how all these components work together in or MDD approach to achieve interoperability within agent platforms and other technologies. First, there are some technical details that need to be addressed, such as the tools and languages used to define and execute the metamodels and mappings. The metamodels presented in Sections 4 and 4.4 were modeled originally in IBM's Rational Software Modeler and the exported to Ecore, the metamodel part of the Eclipse Modeling Framework (EMF) [38]. Ecore represents the meta-metamodel on which our approach is based. Furthermore, the PIM4SOA metamodel is also available in Ecore. For defining and executing the model-to-model transformations, the Atlas Transformation Language (ATL) [39, 40] was chosen, since it offers a series of plugins and tools for the Eclipse Framework and supports EMF as source and target language, among many others. Once the model-to-model transformations have been performed, the produced PSMs must be serialized to the particular programming language, i.e. JACKMM models are transformed to JACK Gcode whereas JADEMM models are directly transformed to Java. In both cases the serialization is implemented using the MOFScript language [41], which is currently a candidate in the OMG RFP process on MOF Model-to-Text transformation.
In MOFScript a set of serialization rules (i.e. templates) is created following the structure of the source MOF-based metamodel, i.e. JACKMM or JADEMM. This means that the information regarding the concept itself as well as the references to other concepts are extracted and assigned to the template's attributes.
For the serialization of JACKMM models, we create a template for the concepts Event, Role, Capability, NamedData, NamedRole, Agent, Plan, Team, NamedData and TeamPlan. For each instance of the mentioned concepts in the JACKMM model, a new file is generated. For example, for each Team instance in the JACKMM Model the template creates a new file with the extension *gteam*. Beside the templates for the main concepts, we create a template that generates a project file that contains a reference to all newly created JACK files. By importing the project file into the JACK development IDE, we imported all the other JACK files that could now be compiled to generate Java code that could execute the JACKMM model.
For the serialization of JADEMM models, there was a possibility of using the EMF generated Java interfaces and implementation classes as serialization. However, some issues were found. Since Java does not support multiple inheritance and JADE requires that the instance extends from their own model—for example Agents should extend from jade.core.Agent, concepts that inherit from other concepts in the metamodels are not able to extend both an EMF class and a JADE class at the same time. Additionally, the EMF property *instanceClassName* that would allow an EMF class to be linked to a Java class, is actually taken as a superinterface to the interface that represents the desired concept. Given these issues, as previously mentioned, a template-based MOFScript serialization was chosen to generate the Java code. Once this classes are generated, they only need to be compiled and executed with the JADE libraries loaded in the classpath.

11 Discussion

This paper presented a platform-independent model for agents together with a MDD approach to develop MAS. MDD can be considered as new paradigm to develop software systems as the different stages with the software development process can be connected by defining mappings. In the context of agent-oriented software engineering, we have identified the following advantages that our approach offers:

- The PIM4Agents defines an abstract language specifying a concrete syntax to design and model agent systems. Furthermore, by defining model transformations from PIM to PSM we

could provide a straightforward interface to implement the generated PIM4Agents models and thus we decreased the knowledge that is required to implement MAS with respect to technical details of agent architectures and MAS development tools.
- MDD addresses interoperability issues between agent-oriented systems and other fields of applications (e.g. Peer-to-Peer systems, Web services and service-oriented architectures (SOA)). In particular, when having an application-oriented metamodel in accordance to Ecore as meta-metamodel, we can easily define mappings to the PIM4Agents metamodel and use the already existing vertical transformations to execute the application with JADE or JACK. In this paper, we have discussed the realization basing on a metamodel for SOAs.
- The presented vertical and horizontal mappings show that it is possible to have interoperability within different agent systems and technologies that are compliant/generated with a model definition.

12 Conclusion

This paper presents a platform-independent model for agents (called PIM4Agents) that specifies a clear syntax and semantic that defines how to develop agent systems. We described the core concepts of the PIM4Agents in detail and discussed how this metamodel could be used in a MDD scenario to simply the generation of executable agent systems.
The PIM4Agents is divided into four viewpoints, i.e. agent viewpoint, organization viewpoint, interaction viewpoint and behavioural viewpoint that allow to model the core characteristics of agent systems.
Furthermore, the metamodels for JACK and JADE—which could be considered as platform-specific frameworks to develop agent systems—were discussed. On their base vertical transformations from the PIM4Agents to JACK and JADE were defined that allow to provide a straightforward interface for implementation as the abstract descriptions basing on the PIM4Agents language could be easily used to generate executable code.
Additionally, we described how to transfer service-oriented architectures—as one feasible application area—to the PIM4Agents. Therefore, we illustrated (i) a platform-independent model for SOA (PIM4SOA) and (ii) how the concepts of the PIM4SOA can be transformed to agent-oriented concepts described by the PIM4Agents (horizontal mappings). This model description in accordance to the PIM4Agents then again be transformed to executable code by applying the vertical mappings.

References

1. 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 (June 2003)
2. D'Souza, D.: Model-Driven Architecture and Integration - Opportunities and Challenges, Version 1.1, Kineticum. (2001)
3. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Core Specification, Document ptc/04-10-15, October 2004, http://www.omg.org/docs/ptc/04-10-15.pdf (October 2004)
4. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Document ptc/05-11-01, November 2005, http://www.omg.org/docs/ptc/05-11-01.pdf (November 2005)
5. Bellifemine, F., Poggi, A., Rimassa, G.: JADE - a FIPA-compliant agent framework. In: Proceedings of the Practical Applications of Intelligent Agents. (1999)
6. AOS: JACK Intelligent Agents, The Agent Oriented Software Group (AOS), http://www.agent-software.com/shared/home/ (2006)
7. Trencansky, I., Cervenka, R.: Agent modeling language (AML): A comprehensive approach to modeling mas. Informatica **29**(4) (2005) 391–400
8. Bauer, B., Müller, J., Odell, J.: Agent UML: A formalism for specifying multiagent interaction. In: Agent-Oriented Software Engineering: First International Workshop, AOSE 2000. Lecture Notes in Computer Science 1957, Springer-Verlag (2001) 91103
9. Bauer, B.: UML Class Diagrams revisited in the context of agent-based systems. In: Agent-Oriented Software Engineering II: Second International Workshop, AOSE 2001. Lecture Notesin Computer Science 2222, Springer-Verlag (2002) 101118

10. Ferber, J., Gutknecht, O.: 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 (ICMAS'98). (1998) 128–135
11. 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)
12. Picard, G., Gleizes, M.P.: 8, The ADELFE Methodology. In: Methodologies and Software Engineering for Agent Systems, The Agent-Oriented Software Engineering Handbook. Kluwer Academic Publishers (2004)
13. Zambonelli, F., Jennings, N., Wooldridge, M.: Developing multiagent systems: the Gaia methodology. ACM Transactions on Software Engineering and Methodology **12**(3) (2003) 417–470
14. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems **3**(3) (2000) 285–312
15. Pavn, J., Gmez-Sanz, J.: Agent oriented software engineering with INGENIAS. In: Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003. Lecture Notes in Computer Science 2691, Springer-Verlag (2003) 394–403
16. Cossentino, M.: From requirements to code with the PASSI methodology. In Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies, Hershey, PA, USA, Idea Group Inc. (2005)
17. Serrano, J.M., Ossowski, S.: On the impact of agent communication languages on the implementation of agent systems. In: Proceedings of the Eight International Workshop CIA 2004 on Cooperative Information Agents. Volume 3191 of Lecture Notes in Computer Science., Berlin et al., Springer (2004) 92–106
18. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In Odell, J., Giorgini, P., Müller, J., eds.: Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004. Revised Selected Papers. Lecture Notes in Computer Science 3382, Springer-Verlag (2005) 62–77
19. Bratman, M.E.: Intentions, Plans, and Practical Reason, Cambridge, MA (1987)
20. Huber, M.J.: JAM: a BDI-theoretic mobile agent architecture. In: Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, USA (1999) 236– 243
21. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desire-intention model of agency. In: Proceedings of Agents, Theories, Architectures and Languages. (1999)
22. Amor, M., Fuentes, L., Vallecillo, A.: Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In: Agent-Oriented Software Engineering (AOSE-2004). Number 3382 in Lecture Notes in Computer Science (2004) 93–108
23. Guessoum, Z.: MAS Meta-Models and MDA, AgentLink III AOSE TFG2. Online at: http://www.pa.icar.cnr.it/c̃ossentino/al3tf2/docs/ zahia_slovenia.pdf (2005)
24. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In: Proceedings of the 5th International Workshop on Agent-Oriented Software Engineering (AOSE 2004). Number 3382 in Lecture Notes in Computer Science, Berlin et al., Springer (2005) 62–77
25. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: Model Driven Development of Multi-Agent Systems. In Rensink, A., Warmer, J., eds.: ECMDA-FA. Volume 4066 of Lecture Notes in Computer Science., Springer (2006) 284–298
26. Moraitis, P., Spanoudakis, N.I.: The Gaia2Jade Process for Multi-Agent Systems Development. Applied Artificial Intelligence **20**(2-4) (2006) 251–273
27. Beydoun, G., Gonzalez-Perez, C., Low, G., Henderson-Sellers, B.: Synthesis of a generic MAS metamodel. In: SELMAS '05: Proceedings of the fourth international workshop on Software engineering for large-scale multi-agent systems, New York, NY, USA, ACM Press (2005) 1–5
28. Zambonelli, F., Jennings, N., Wooldridge, M.: Organizational rules as an abstraction for the analysis and design of multi-agent systems. International Journal of Software Engineering and Knowledge Engineering **11** (2001) 303–328
29. Davis, R., Smith, R.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence **20** (1983) 63 109
30. 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) 312–319
31. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In Giunchiglia, F., Odell, J., Weiß, G., eds.: Agent-Oriented Software Engineering (AOSE-2002). Volume 2585 of Lecture Notes in Computer Science., Berlin et al., Springer (2002) 174–185

32. Cervenka, R., Trencanský, I., Calisti, M., Greenwood, D.A.P.: AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling. In: Agent-Oriented Software Engineering (AOSE-2004). Number 3382 in Lecture Notes in Computer Science 3382, Berlin et al., Springer (2004) 31–46
33. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. In: Agent-Oriented Software Engineering (AOSE-2000), Berlin et al., Springer (2001) 91–103
34. 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.: International Conference on Autonomous Agents and Mulitagent Systems (AAMAS-05), ACM (2005) 1121–1122
35. JADE project: JADE Application Programmer Interface 3.4.1 (November 2006)
36. FIPA: FIPA ACL Message Structure Specification (fipa00061). FIPA. (2001)
37. 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)
38. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Framework. Addison Wesley Professional (2003)
39. ATLAS Group, INRIA & LINA, University of Nantes: INRIA, ATL - The Atlas Transformation Language Home Page, http://www.sciences.univ-nantes.fr/lina/atl/ (2006)
40. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: MoDELS 2005, Montego Bay, Jamaica. (2005)
41. SINTEF ICT: MOFScript, http://www.eclipse.org/gmt/mofscript (2006)

A Platform-Independent Model for Agents

Christian Hahn, Cristián Madrigal-Mora and Klaus Fischer

RR-07-01
Research Report