# Model Driven Design of Multiagent Systems

Klaus Fischer, Stefan Warwas, and Ingo Zinnikus

German Research Center for Artificial Intelligence (DFKI) GmbH
Saarbrcken, Germany
{Klaus.Fischer,Stefan.Warwas,Ingo.Zinnikus}@dfki.de

**Abstract.** In general software engineering modelling of software systems had a significant impact on the manner in which complex systems are designed. The Model Driven Architecture (MDA) proposed by the Object Management Group (OMG) provides a formal framework that allows to define dedicated modelling languages for different application domains. Already in the model driven design of service-oriented architectures one can identify concepts that are common in the design of such systems and what agent-based systems concerns. To directly use the MDA framework for the design of multiagent system (MAS) is therefore an obvious step. In this article we advocate the domain specific modelling language DSML4MAS for modelling MAS. However, our aim is not to just define the language, we propose a framework for DSML4MAS that allows its adaptation and dynamic development in the future. Our vision is that in the near future model repositories for model fragments that can be flexibly combined will be established and propose basic concepts that can support the development of MAS in this context. The interaction aspect is especially important in MAS design and one of the most obvious aspects where model exchange and model re-use is highly desirable. The article therefore presents the interaction aspect in more details and discusses the features that are available in the DSML4MAS.

## 1 Motivation

With the success of service-oriented architectures and the ever-growing connectivity of applications in the Internet, agent technologies are becoming even more attractive than they were in the past. However, many times the system design not only in agent-oriented applications is tightly bound to the execution environment. Although we are far from a state where system engineers would not care about the technologies deployed in the execution environment, it is clear that it would be highly desirable to be able to separate system design from such technologies and with this make it more sustainable regarding the evolution of software concepts. The model driven architecture proposed by the Object Management Group (OMG[1]) provides the basic ideas that can significantly improve the possibilities to maintain system designs while new technologies emerge or

[1] www.omg.org

J. Filipe and A. Fred (Eds.): ICAART 2011, CCIS 271, pp. 11–27, 2013.

already available technologies get adapted to new requirements. Agent technologies can contribute in this enterprise because they provide helpful abstractions for the design of complex systems.

In the following we present the ingredients of our approach. Section 2 gives an overview of the overall approach. In Section 3 we present details of the PIM4Agents metamodel that forms the core of DSML4MAS. We zoom in on the interaction aspect of PIM4Agents in Section 4 because this aspect is one of the most obvious where exchange of models and model fragment among system engineers is desirable. Section 5 presents uses cases in which we evaluate our approach and an illustrative example of the use of a concrete interaction model. Section 6 gives some pointers to related work and Section 7 draws conclusions and directions for future research.

## 2 Framework for Model Driven Design of Multiagent Systems

In this article we adopt a model driven approach to the design of agent-based systems. The basic ideas of the approach were developed in the EC[2] funded research projects ATHENA[3] and SHAPE[4] and are now further developed in the EC funded research project COIN[5]. The main achievement in this approach is the definition of the domain specific modelling language DSML4MAS[6] [7]. The metamodel PIM4Agents forms the core of DSML4MAS. From PIM4Agents we derive our modeling tool which is built on the Eclipse EMF/GMF technology stack [19]. However, we do not only aim at just providing the modelling language and tool support. What we want to come up with is a framework that allows to extend and refine the core metamodel by additional or more specialized concepts. For this the metamodel is separated into different parts that deal with specific concerns of the design of a multiagent system. We refer to these parts of the metamodel that form separate meaningful entities with the term *aspect*. The idea is to provide a framework that allows to specifically design and flexibly adapt the different aspects. This approach allows to extend the core of the metamodel by pluging in different realizations of the foreseen aspects. We further distinguish between the aspects into which the metamodel is separated and the different viewpoints that are supported by the modelling tool. A viewpoint in the modelling tool is defined by a diagram that displays a collection of concepts and how they relate to each other. Additionally, a tool box that allows to manipulate the concepts in the diagram is provided, e.g. add or delete new instances of a specific concept or introduce additional relations. The overall goal is to allow both a flexible definition of the aspects in the metamodel as well as a flexible definition of viewpoints in the modelling tool.

[2] European Commission.
[3] `http://www.modelbased.net/aif/`
[4] `http://www.shape-project.eu/`
[5] `http://www.coin-ip.eu/`
[6] `http://sourceforge.net/projects/dsml4mas/`

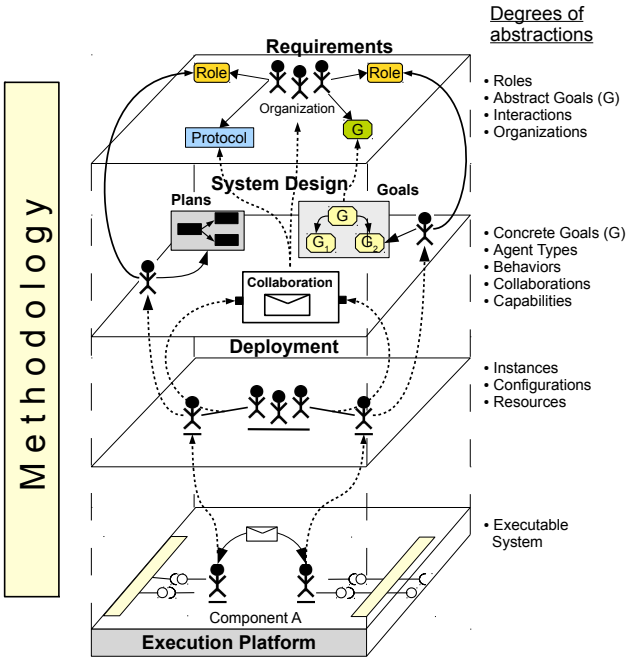


**Fig. 1.** Framework overview

As already mentioned, the core of our model-driven framework for developing multiagent systems is the PIM4Agents metamodel. PIM4Agents is independent of a concrete execution platform but inherently possesses different degrees of abstraction (see Figure 1). The requirements layer is the most abstract degree and covers abstract goals, roles, interactions, and organizations. The system design degree contains (i) agent types, (ii) behavior templates, (iii) concrete goals, etc. The lowest degree is the deployment layer which specifies concrete deployment configurations (e.g. agent instances and resources).

Our aim is to define for DSML4MAS a plugin framework that allows to flexibly extend or completely replace the different foreseen aspects. Additionally to the idea that parts of the metamodel can be extended by plugins with different realizations, we assume that there will be a landscape of metamodels which share a common sub-set of concepts. It is a quite safe guess that it should be possible to arrange these different metamodels in a hierarchy of specialization with a common root. At least the concept of an agent is likely to be part of any metamodel that people come up with when they want to do modelling of agent-based or multiagent systems. With a landscape of metamodels in mind, it is easy to foresee also a landscape of model repositories that hold models or model fragments according to different metamodels. In this framework collaborative modelling can be supported in the sense that system engineers can store and retrieve models or model fragments to and from model repositories.

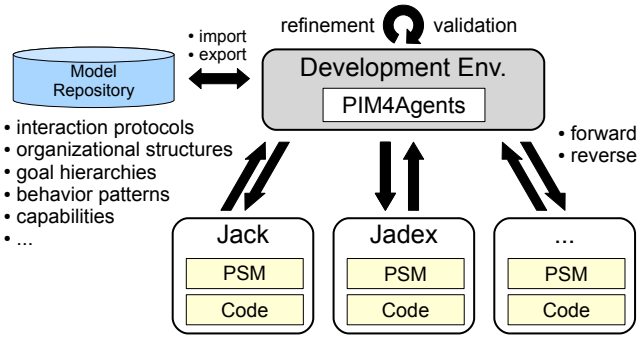


**Fig. 2.** Overview of the development environment

Theoretical results (e.g. about the efficiency of a specific model for a specific auction mechanism) can be directly linked to such models or model fragments. Model transformations can be used to transform model instances from one metamodel to another one or into different execution environments as well as to merge together model fragments from different sources. Figure 2 shows the basic setup for the DSML4MAS development environment.

## 3 The PIM4Agents Metamodel

The metamodel PIM4Agents forms the core of the domain specific modelling language DSML4MAS. The metamodel is structured into different aspects. An aspect contains a collection of concepts and definitions how these concepts relate to each other. OCL constraints are used to express semantics. There are two types of OCL constraints: (i) constraints to compute derived information, i.e. the value of a specific attribute is derived from values of other concepts and (ii) integrity constraints, e.g. the value of an integer attribute must be higher than 0 or below a given number.

PIM4Agents supports 12 aspects:

**Multiagent System.** Describes all basic components the MAS is composed of.

**Agent.** Describes single autonomous entities, the capabilities they have to solve tasks and their roles they play within the MAS.

**Organization.** Describes how single autonomous entities cooperate within the MAS and how complex organizational structures can be defined.

**Role.** Definines the requirements an agent should fulfull when it wants to engage in an organizational structure.

**Interaction.** Describes how the interaction between autonomous entities or organizations takes place.

**Behavior.** Describes how plans are composed by complex control structures and simple atomic tasks.

**Information.** Contains any kind of resource that is dynamically created, shared, or used by agents or organizations.

**Deployment.** Allows to define a MAS at instance level that can be used for startup.

**Goal.** Explicit representation of a goal hierarchy in form of an and/or tree representation.

**Event.** A stimulus the agents can react to.

**Environment.** Allows the agents to sense information from the outside environment and to manipulate the outside environment with actuators.

**Resource.** Was introduced to connect agents with a service-oriented environment.

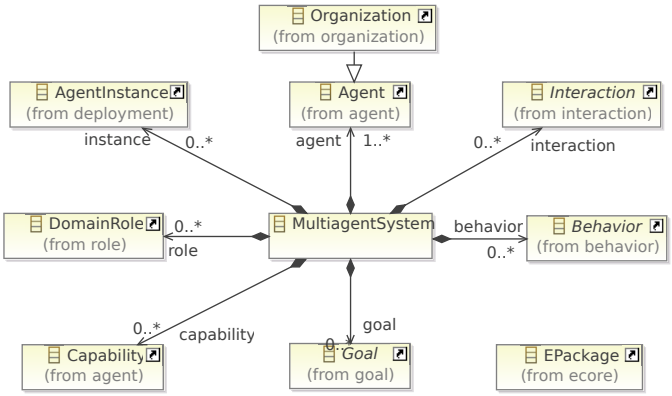


**Fig. 3.** The Multiagent System Aspect in PIM4Agents

Fig. 3 shows the MAS aspect which includes all major concepts a MAS is composed of. Out of these we want to briefly discuss the following: Agent, Organization, Role, Interaction, and Behavior.

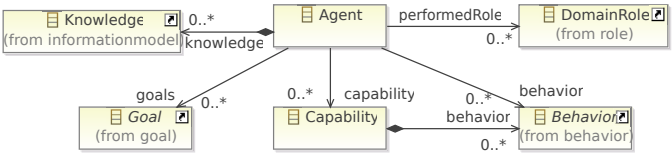


**Fig. 4.** The Agent Aspect of PIM4Agents

An agent (see 4) has behaviors that may be grouped to capabilities which together with the information in the information model allow the agent to achieve its goals. The agent might perform roles in an organizational structure.

An organization (see 5) is in the first place composed of roles. The agents that actually perform these roles are grouped together in concrete collaborations to be able to achieve the organization's goals. DomainRoles allow a renaming of role

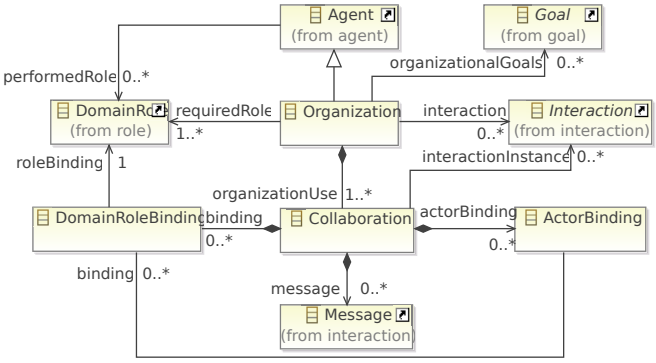


**Fig. 5.** The Organization Aspect of PIM4Agents

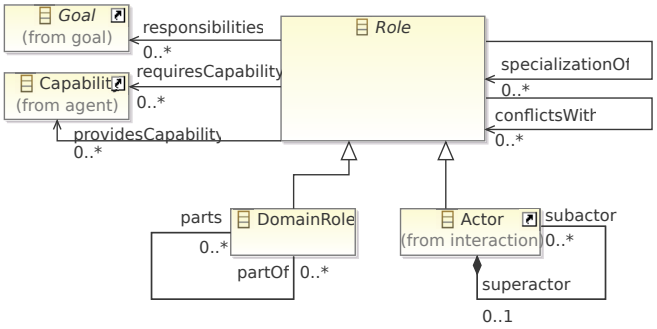


**Fig. 6.** The Role Aspect of PIM4Agents

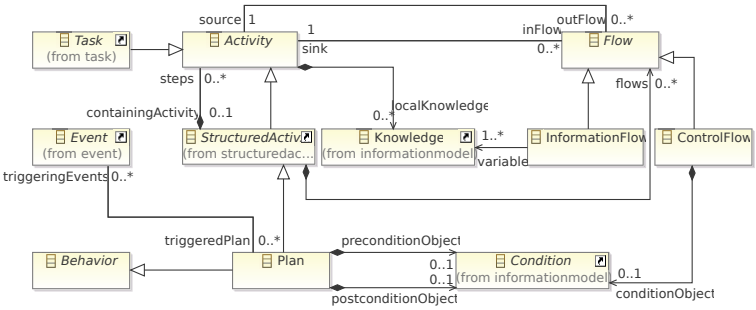


**Fig. 7.** The Behavior Aspect of PIM4Agents

types of the organization in collaborations. Actor bindings bind the DomainRoles to the interactions which are used in the collaboration to prescribe how an interaction has to be performed when the involved agents work towards a given goal.

Roles (see 6) are responsible for specific goals and require capabilities from the agents performing the goals that allow the agents to actually successfully achieve the goals the role is responsible for. Roles can be specialized and can be in conflict with each other, meaning that an agent that performs one role might not be allowed to perform a specific other role. DomainRoles and Actors are specialization of the more general role concept.

Figure 7 displays the Behavior aspect of the PIM4Agents metamodel. A behavior for an agent is specified by a plan. Each plan is composed of a trigger event, a pre-condition, a post-condition, and a set of activities. Activities can be complex patterns like for examples loops or simple tasks. Flows provide a sequencing among the activities. Special tasks (i.e. BeginTask and EndTask) mark for a diagram where the execution of the plan starts and where it ends. The intended semantics is that when an event arises within the agent's body that matches the trigger event, the preconditionObject is checked. If this Object returns with true, the execution of the body (i.e. the set of activities) is started at the BeginTask. Execution of the plan body ends when an EndTask is encountered. It is assumed that the postconditionObject evaluates to true when the execution of the plan body terminates.

## 4 Taking a Closer Look at Interactions

Regarding the design of agent interactions we take for the discussion in the article a restricted point of view. We purely concentrate on what we call contract-based interactions. This means that we assume that for the interactions which we want to take into account a predefined contract exists which the agents use when performing the interaction. The definition and analysis of interaction protocols like the Contract Net Protocol (see below) is complex and therefore it is beneficial if system engineers can flexibly adopt interaction protocols that are well understood. The approach presented in this article allows to set up repositories for model fragments (e.g. interaction protocols) and adopt them in separate designs of multiagent systems by transforming the model fragments into representations that can be directly included into the local design (e.g. capability specifications of individual agents).

To make the discussion in this article not too complicate we assume that all models are defined at design time of the system and then purely used at run time. System dynamics is purely restricted to instance level and does not include type level. However, this still allows dynamic assignment of agent instances to roles. How this role assignment is actually done is out of the scope of this article. We assume that an agent that was assigned to take a specific role on the one hand will always try its best to fulfil the obligations the role is asking for and also provides in principle all capabilities the role is asking for. At run time no explicit checks are done on whether such requirements are actually met. We assume that if such checks would be done they would be performed at type level at design time.

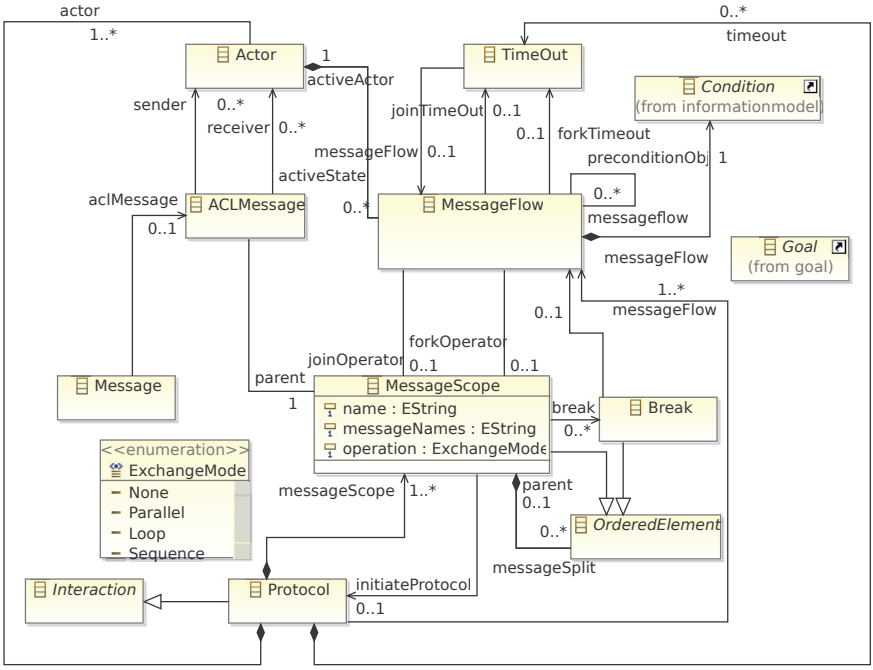


**Fig. 8.** The Interaction Aspect of PIM4Agents

Regarding the metamodel definitions the Interaction aspect (see Figure 8) is one of the most complex in the PIM4Agents metamodel. However, the most important concepts to understand are: Protocol, Actor, MessageFlow, and MessageScope. The protocol is actually the core concept that defines an interaction. The reason to not directly replace the concept of interaction with the concept protocol is that protocols are intuitively understood as message based interactions and we want to be open to allow in future work other forms of interaction, too.

A protocol is in the first place composed of a set of actors. For each actor a set of message flows defines the different states in which the interaction could end up. Message scopes connect two message flows of two different actors. An attribute tells whether the message flow has a fork operator (i.e. it sends a message) or a join operator which means that the message flow receives a message. The message scope refers to the concrete message that is sent from the sender (actor) to the receiver (actor). In each actor exactly one message flow is marked as initial message flow which means that the protocol execution will start for this actor in the state that is defined by this message flow. However, in the whole protocol there is only one actor for which the initial message flow has a forkOperator (i.e. the initial message flow sends a message). We call this message flow start message flow of the protocol.

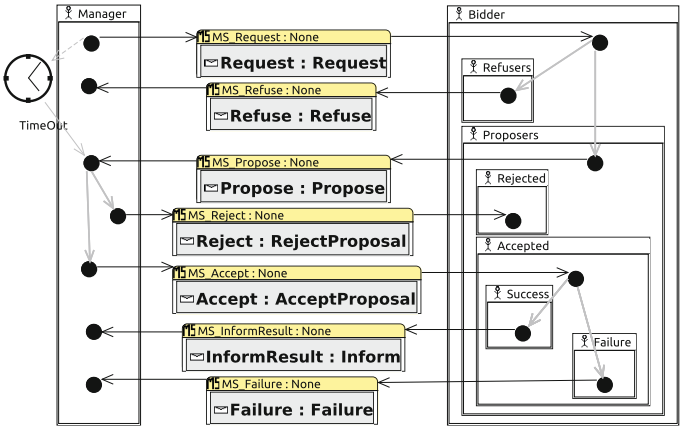


**Fig. 9.** The Contract Net Protocol

The core part of the contract the agents use for interaction is given with a protocol. Figure 9 shows the diagram for the Contract Net Protocol[7] [5]. In this protocol a manager agent tries to find an agent in a group of bidders that is selected to provide a specific service for the manager. We present some more details on the usefulness of this protocol in Section 5. The building blocks for protocols are (i) actors, which might be separated into sub-actors, (ii) message flows, and (ii) and messages. Actors refer to the participants of the protocols. Role bindings are used to define the requirements to agent types that actually could take the part of a specific actor in a protocol. Message flows mark specific states in the protocol execution. If a message flow has more then one exiting arc exactly one of these arcs can be chosen to continue protocol execution, which means that the outgoing arcs have an xor semantics. An additional assumption is that each message flow that receives a message spawns of an achieve goal event, where the abstract goal's name is derived from the name of the incoming message which is by definition unique, i.e. a message flow can receive at most one message. To achieve such an abstract goal might turn out to be a complex process within the agent and might very well involve the interaction with other agents which are then, behind the scenes, again organized by using contracts. Messages are defined by message types.

With these conventions the behavior that an agent needs to comply to when it engages in a specific interaction is defined in the protocol description. However, the protocol specifies the communication behavior only. All capabilities that are available from the agent's body are addressed by spawning off achieve goal events and the only direct body capability the protocol itself relies on is that the agent is able to send the messages according to the specified message types. In this

[7] Please do not confuse the name of the Contract Net Protocol with what we call contract based communciation. The dual use of the term *contract* is basically pure chance.

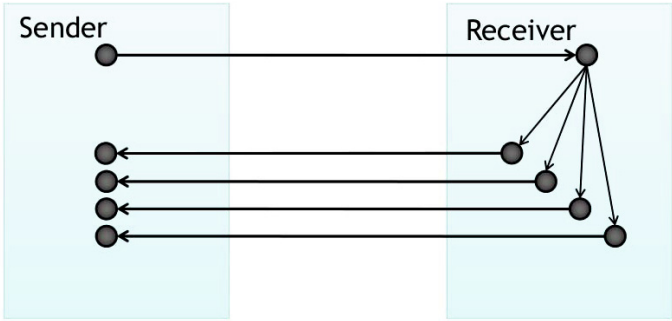


**Fig. 10.** Basic Communication Pattern

sense the specification of the interaction among the agents is a choreography of the capabilities of the agents that are involved in the interaction.

When it comes to executing the protocol at run time one has at least two options to choose from. In the first option the interaction protocol would be directly interpreted by a protocol interpreter that is included in the agents' bodies. In the second option the protocol is transformed into local behaviors for each of the participating agents which can be directly executed in some execution environment. The first option is more flexible but the second option is easier to implement. For this reason we use the second option. This has also the advantage that the local behavior can be produced with a model-to-model transformation at the PIM4Agents level which means that the resulting behaviors can be transformed into all different execution environment for which a transformation of PIM4Agents models is available. We therefore do not interpret the protocol model directly but transform the protocol model at design time into capabilities that provide the respective communication behavior that is required by the contract to which the protocol belongs. To achieve this a separate capability is generated from the protocol model for each of the given actors. We can identify a basic pattern which allows to already design a large number of different protocols (at least regarding those which are explicitly represented by models at design time). This basic pattern is displayed in Fig. 10. It always starts with one actor sending a message to another actor and then waiting for all answers to this message.

Behind all actors of a protocol any number of agent instances might be hiding except for the actor that contains the start message flow. Only one individual agent is allowed to play the role that is connected to this actor. Only the start message flow actually sends multiple messages to all agent instances hiding behind the actor that receives this message. All subsequent messages are exchanged in a bilateral manner. However, this means that the start message flow spans off a set of parallel interaction threads. For some protocols it is necessary to synchronize (see Fig. 11) these interaction threads. For example in the Contract Net Protocol ([5] see Fig. 9) the manager has to wait till all bidders have replied

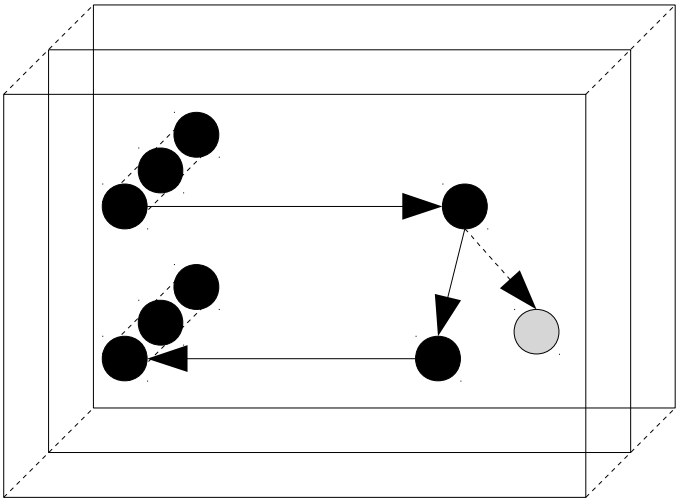

**Fig. 11.** Synchronization between parallel execution threads

to the call for proposals or some specified deadline has passed. Only then the protocol can proceed and the best bidder be selected. To define this kind of synchronization at the modelling level we provide the following concepts:

- For each protocol instance in each actor we maintain a context in which the state of the ongoing protocol execution can be maintained.
- In the protocol context we maintain a table which allows us to find for each message sent all replies that have been received so far.
- Message flows can be marked as synchronized. A condition specifies what needs to happen before the synchronization is successfully achieved. Such a condition can be that a specific number of replies has been received or that a specific event (e.g. a timeout) has occurred. All replies to a sent message that are received after the synchronization was successfully achieved are ignored.
- A special variable MaxMessages in the protocol context gives the number of how many messages were actually sent in the start message flow.

With this basic machinery it is already possible to design a large number of complex protocols. A limitation of the presented concepts is that it is not possible to model protocols where messages are sent out to more than one actor while the sender waits for the replies to these messages in a concurrent manner. However, because several agent instances can hide behind an actor the only limitation is that sending concurrent messages to more than one group of agents hiding behind one of the actors in the protocol diagram is not possible in one model. However, because each of the agent instances at any specific stage of the protocol execution (i.e. a specific message flow), where it receives a message, spawns off an achieve goal event to produce the message that should be sent next, in the process of achieving this event can of course again initiate the execution of an

interaction protocol. So although this interaction is not visible in the original protocol diagram, any number of cascading protocol executions can result from the execution of a specific interaction protocol. To restrict sending of concurrent messages to the start message flow of an interaction protocol is therefore actually not a real restriction but enforces structure that reduces complexity.

The interaction protocol describes the interaction among the actors (i.e. the agents that perform the roles that are bound to the actors) from a centralized point of view.

To actually describe the model to model transformations from interaction protocols to local behaviors for individual agents we use operational QVT[8].

```
helper pim4agents::interaction::Actor::collectMsfs () :
                    Set(pim4agents::interaction::MessageFlow) {
    var res : Set(pim4agents::interaction::MessageFlow);
    res := self.activeState;
    self.subactor->forEach(a) {
        res := res->union(a.collectMsfs());
    };
    return res
}
```

**Fig. 12.** Helper of a QVT Transformation

Operational QVT offers two types of procedural concepts. The first straight forward concept are helpers. Helpers are very similar to methods of an object oriented language. The helper can be typed and in this case it basically extends the signature of the concept that is defined in the metamodel and allows computations that are useful for handling this concept. The definition of the helper also shows how the concepts in the PIM4Agents metamodel are addressed: first the metamodel is named, then the package in the metamodel, and last the concept. In the body of the helper OCL expressions can be used which allow to express complex computations in a compact manner. The helper in Figure 12 collects all message flows that are included in an actor in a given interaction protocol.

```
mapping PIM4Agents::interaction::Actor::toDomainRole () :
                                    PIM4Agents::role::DomainRole {
  var msf : Set(pim4agents::interaction::MessageFlow) :=
       self.collectMsfs();
  var rmsf : Set(pim4agents::interaction::MessageFlow) :=
       msf -> select(d|d.isInitialMessageFlow or
       ((d.forkOperator <> null) and (d.MsfSuccessors(msf)->size() > 0)));
  name := 'Role' + self.name;
  providesCapability := rmsf.map toCapability(msf,rmsf);
}
```

**Fig. 13.** QVT Mapping Rule, Creation of a Domain Role from an Actor of a Protocol

[8] Query view transformation.

The second procedural concept that operational QVT offers are mappings (see Figure 13). Mappings look quite similar to helpers, however, there is an important difference. Mappings result in a link between the entity they are applied to and the entities they create. This means that a mapping can be called several times but the structures it creates are only created once and that the same structure can be mapped to different places (i.e. attributes of concepts) in the model instance that is produced as a result of the QVT transformation. In the mapping the variable msf holds all MessageFlows that are contained in the actor and rmsf is the subset of MessageFlows in the set msf that are considered relevant. Relevant MessageFlows are those where a message is sent and an answer is expected. The condition d.forkOperator <> null says that d sends a message and the helper d.MsfSuccessor(msf) returns the set of message flows that receive an answer for the message sent in d. This exactly corresponds to the situation of the actor Sender in the communication pattern shown in Figure 10. The MessageFlow sending the message would be considered relevant. The four MessageFlows in the lower part would be returned by the helper MsfSuccessors. If the communication pattern is only used once in the actors the sending MessageFlow in the actor Sender and the receiving MessageFlow in the Receiver would be marked as initial message flows where the former is called the start MessageFlow.

## 5 Use Cases

The presented approach for the design of multiagent systems is currently further investigated and practically used in the research projects ISReal and COIN.

The COIN project investigates collaboration and interoperability for networked enterprises. In this context we use the modelling approach presented in this article for the design of negotiation processes in enterprise systems. Negotiations occur prominently in business interactions between competing partners, but also between cooperating partners, e.g. the participants in supply chains or *virtual enterprises*. One scenario we are looking at in this work is the situation in which a production plan for collaborating partners in a supply chain has already been scheduled. In this setting there are two scenarios for which additional negotiations could be necessary while a production plan is executed: (i) for a specific step the service provider (e.g. a transportation service or a supplier of raw material) has been left open or (ii) it turns out that a pre-negotiated service provider cannot provide the agreed service. For both scenarios the Contract Net Protocol (see Fig. 9) can be used to organize the negotiation. Services the manager agent can chose from are registered and published in a *general service platform (GSP)* which provides discovery and invocation support. Services are semantically annotated using the WSMO/WSML language which facilitates ad-hoc service provisioning and execution.

In COIN, service provision can be supported at design time and/or at run time. In the first case, a process modeler can check for available services while designing the interactions and agent plans. In the second case, when a service

provider drops out or e.g. cannot fulfill the required quality of service, a new service provider can be determined by retrieving a list of candidate services from the GSP and selecting the best service using the Contract Net Protocol.

The focus of ISReal is the design of agents and multiagent systems in a virtual reality settings where the agents represented by avatars that form their virtual bodies. Digital factories are one of the application areas ISReal is aiming at. The aim of the ISReal project is to develop an execution platform for semantic 3D simulations [8]. The basic idea of ISReal is to add semantic descriptions to 3D objects and specify their functionality by semantic service descriptions. Our approach is based on the semantic Web standards OWL[9], OWL-S[10], and RDFA[11]. Agents perceive the annotated facts and service descriptions and use them for reasoning and planning. The scene runs in a 3D-enabled Web browser based on XML3D[12]. We use our model-driven development environment DDE for engineering the agents that control the avatars (their virtual body) in the 3D scene. Figure 14 depicts the application of an agent interaction protocol in a 3D simulation. Agent *A1*'s target is to buy ingredients for the production of some pills on the pill filling machine shown in the background. The Contract Net Protocol is used by agent *A1* to negotiate with the pharmacy agents *A3*, *A4*, and *A5*. In Figure 14, agent *A5* won the auction.

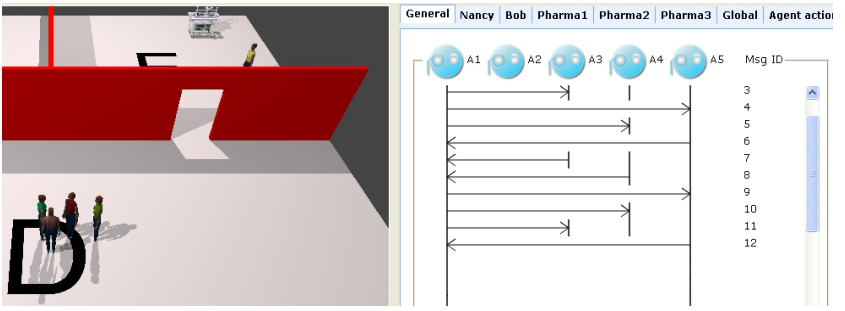


**Fig. 14.** Contract Net example in the ISReal scenario

## 6 Related Work

Communication is an important aspect of agent-based and multiagent systems and therefore has been intensively investigated. The FIPA[13] initiative was originally founded to produce specifications for software standards for heterogeneous and interacting agents and agent-based systems. One of the main achievements of

[9] http://www.w3.org/2004/OWL/
[10] http://www.daml.org/services/owl-s/1.2/
[11] http://www.w3.org/TR/rdfa-syntax/
[12] http://www.xml3d.org/
[13] www.fipa.org

FIPA was a standard proposal for an agent communication language (ACL). [10] gives and overview of FIPA and related activities. Although FIPA also made proposals for standards for communication protocols[14], execution of such protocols was rather neglected. In his seminal work [16] investigated formal approaches to protocol design. Declarative methods to describe protocols have the charm that the formalisms seem to be clean. However, at least in some cases it is a problem to find out whether protocols can be enacted [6]. More pragmatic approaches do not face this problems because protocol execution is directly foreseen, however, conformance to prescribed behavior is a general problem.

While the areas of general software engineering and agent-oriented software engineering lived for years without much interaction, in recent years the concepts of the two areas have grown significantly together. A good example for the common interest is the Agent Platform Special Interest Group of the Object Management Group (OMG) which has the goal to foster OMG specifications in the agent area. Other examples are the Agent Modelling Language (AML) [18] a semi-formal visual modeling language for the definition, modeling, and documentation of systems that adopt agent technologies. AML is defined as an extension of the Unified Modeling Language (UML[15]) using the most important OMG frameworks. Agent UML (AUML) [2] extended UML sequence diagrams with interaction protocols. Besides agent-based modeling approaches several methodologies were proposed (e.g. Tropos [15], Prometheus [14,17]) that provide mechanisms to support the specification, the analysis, and development of agent-based systems. Additionally proposals for metamodels for agent-based systems are on the table (e.g. Gaia [20], PASSi [4], and ADELFE [3]).

Protocol projections have been studied already early in the area of verification of communication protocols (e.g. [11]). Recently, these techniques are also used in the context of collaborations [12]. More specifically, projections are used for generating executable business processes and orchestrations from choreography descriptions (e.g. [13], [9]), restricted to languages for business process modeling.

## 7 Conclusions

This article presented a framework for the model driven design of MAS. The framework is built around the domain specific modelling language DSML4MAS. The core of DSML4MAS is defined by the metamodel PIM4Agents that includes platform independent concepts for the design of MAS. PIM4Agents is separated into 12 major aspects for MAS design. The general framework proposes a plugin architecture where these different aspects can be replaced or refined in a flexible manner. In the discussion of the PIM4Agents metamodel the article concentrates on the Interaction aspect because it is an obvious choice when it comes to reuse of model fragments in MAS design. The design of interaction protocols is complex and tedious and therefore reuse of well-understood protocols is highly desirable. The article presents a proposal how the Interaction aspect in DSML4MAS can

[14] `http://www.fipa.org/repository/ips.php3`
[15] `www.uml.org`

be realized. This approach is pragmatic and procedural with the advantage to define an operative semantics for the models in the transformation to a specific execution environment. Further research needs to be done to link the presented approach with proposals for declarative protocol specifications like for example presented in [16,6,1].

The main contribution of the work presented in this article aims at the design of agent-based systems and multiagent systems from a software engineering perspective. In this work AI aspects are not directly obvious. However, in matchmaking of queries on model repositories and in the internal reasoning of the agents AI topics are of course relevant. At least regarding reasoning work on agent technologies (e.g. the agent development tool JACK) it has been shown that modelling and reasoning can be brought together. We plan to integrate these AI aspects more deeply with our approach in future work.

**Acknowledgements.** The paper is based on work performed in the project COIN (EU FP7 Project 216256; www.coin-ip.eu) funded by the European Community within the IST-Programme of the 7th Framework Research Programme. The authors also thank the contribution from other partners in the COIN consortium.

## References

1. Baldoni, M., Baroglio, C., Chopra, A.K., Desai, N., Patti, V., Singh, M.P.: Choice, interoperability, and conformance in interaction protocols and service choreographies. In: AAMAS (2), pp. 843–850 (2009)
2. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Interaction. In: Ciancarini, P., Wooldridge, M.J. (eds.) AOSE 2000. LNCS, vol. 1957, pp. 91–103. Springer, Heidelberg (2001)
3. Bernon, C., Gleizes, M.-P., Peyruqueou, S., Picard, G.: ADELFE: A Methodology for Adaptive Multi-Agent Systems Engineering. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS (LNAI), vol. 2577, pp. 156–169. Springer, Heidelberg (2003)
4. Chella, A., Cossentino, M., Sabatucci, L., Seidita, V.: From passi to agile passi: Tailoring a design process to meet new needs. In: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2004, pp. 471–474. IEEE Computer Society, Washington, DC, USA (2004), http://dx.doi.org/10.1109/IAT.2004.59
5. Davis, R., Smith, R.G.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence 20(1) (1983)
6. Desai, N., Singh, M.P.: On the enactability of business protocols. In: Fox, D., Gomes, C.P. (eds.) AAAI, pp. 1126–1131. AAAI Press (2008)
7. Hahn, C., et al.: A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems 18, 239–266 (2009)
8. Kapahnke, P., Liedtke, P., Nesbigall, S., Warwas, S., Klusch, M.: ISReal: An Open Platform for Semantic-Based 3D Simulations in the 3D Internet. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part II. LNCS, vol. 6497, pp. 161–176. Springer, Heidelberg (2010)

9. Khadka, R., Sapkota, B., Ferreira Pires, L., van Sinderen, M., Jansen, S.: Model-Driven Development of Service Compositions for Enterprise Interoperability. In: van Sinderen, M., Johnson, P. (eds.) IWEI 2011. LNBIP, vol. 76, pp. 177–190. Springer, Heidelberg (2011)
10. Kone, M.T., Shimazu, A., Nakajima, T.: The state of the art in agent communication languages. Knowledge and Information Systems 2(3), 259–284 (2000), http://dx.doi.org/10.1007/PL00013712
11. Lam, S.S., Shankar, A.U.: Protocol verification via projections. IEEE Trans. Software Eng. 10(4), 325–342 (1984)
12. McNeile, A.T.: Protocol contracts with application to choreographed multiparty collaborations. Service Oriented Computing and Applications 4(2), 109–136 (2010)
13. Mendling, J., Hafner, M.: From WS-CDL choreography to BPEL process orchestration. Journal of Enterprise Information Management 21, 525–542 (2005)
14. Padgham, L., Thangarajah, J., Winikoff, M.: Tool support for agent development using the prometheus methodology. In: Proceedings of the Fifth International Conference on Quality Software, QSIC 2005, pp. 383–388. IEEE Computer Society, Washington, DC, USA (2005), http://dx.doi.org/10.1109/QSIC.2005.66
15. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: From stakeholder intentions to software agent implementations. In: Conference on Advanced Information Systems Engineering, pp. 465–479 (2006)
16. Singh, M.P.: Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications. LNCS, vol. 799. Springer, Heidelberg (1994), http://www.csc.ncsu.edu/faculty/mpsingh/books/MAS/
17. Thangarajah, J., Padgham, L.: Prometheus design tool. In: The 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems, pp. 127–128 (2005)
18. Trencansky, I., Cervenka, R.: Agent modeling language (aml): A comprehensive approach to modelling mas. Informatica 29(4), 391–400 (2005)
19. Warwas, S., Hahn, C.: The DSML4MAS development environment. In: Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 1379–1380. IFAAMAS (2009)
20. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology 12(3), 317–370 (2003)