BOCHICA: A MODEL-DRIVEN FRAMEWORK FOR ENGINEERING MULTIAGENT SYSTEMS

Keywords: Agent-oriented Software Engineering, PIM4Agents

Abstract: Modeling real world agent-based systems is a complex endeavour. An ideal domain specific agent modeling language would be tailored to a certain application domain (e.g. virtual worlds) as well as to the target execution environment (e.g. a legacy execution platform). This includes the use of specialized domain concepts, information models, software languages (e.g. query languages for reasoning about an agent's knowledge), as well as custom views and diagrams for designing the system. At the same time it is desirable to reuse application domain independent model artifacts such as interaction protocols (e.g. auction protocols) or goal/plan decompositions of a certain problem domain that already proved their use in other scenarios. Current agent modeling languages cover the core concepts of multiagent systems but are not thought to be customized for a certain application domain. In this paper we propose a model-driven framework which is based on a platform independent core modeling language that can be tailored through several extension interfaces to the user's needs. The common core language leverages the reuse of existing model artifacts and reduces development time and costs for the creation of application domain specific solutions. We evaluated our approach on a distributed semantic web based execution platform for simulated realities.

1 INTRODUCTION

The research field of *Agent-oriented Software Engineering* (AOSE) is concerned with investigating how algorithms and methods developed in the wide area of artificial intelligence can be used for engineering intelligent software agents in a systematic way. AOSE should not be seen in isolation: As it gets increasingly applied in main stream software engineering it is confronted (of course) with typical problems of today's software development such as (i) an increasing number of software frameworks, programming languages, and execution platforms, (ii) shorter development cycles, and (iii) heterogeneous and distributed IT environments. A key to tackle the rapidly growing complexity in software development is abstraction. Higher-level software languages are required to hide the complexity and focus on the design of IT systems. *Model-driven Software Development* (MDSD) is driven by industry needs to deal with complex software systems. The underlying idea of MDSD is to model the *system under considerations* (SUC) on different levels of abstractions and use model transformations to gradually refine them from requirements specification, to system design, and finally concrete code. Several core aspects of MDSD were standardized by the *Object Management Group* (OMG) as *Model-driven Architecture*[1] (MDA).

[1]http://www.omg.org/mda/specs.htm

During the recent years, several approaches for modeling agent-based systems have been proposed. Although we think that the developed modeling languages are a step into the right direction, they have problems with fulfilling a user's need to efficiently model a certain application domain (e.g. virtual worlds or a legacy execution environment). An ideal modeling language would contain, beside the core concepts of multiagent systems, specific concepts, diagrams, tools, etc. for a certain target environment. Moreover, it is desirable to extend the modeling language with new concepts from agent research (e.g new ways of modeling behaviors or interaction protocols). In this paper we propose a model-driven framework for engineering agent-based systems to overcome the mentioned limitations. Our framework, called BOCHICA, is based on a core domain specific language (DSL) which covers the main aspects of multiagent systems (see Figure 1). The framework provides several interfaces for customizing the language with new concepts, 3rd party software languages, and custom diagrams. We envision the framework as a bridge between agent research and concrete software development. The customizations allow the user to define the right level of abstraction for his application domain without loosing the integration into a larger framework which enables the exchange of model artifacts such as goal hierarchies and interaction protocols.

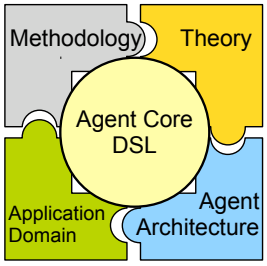

Figure 1: The BOCHICA framework for AOSE is based on a platform independent core DSL and is able to integrate research results, methodologies, and application domains.

The first part of this paper provides background on previous work (Section 2), introduces the general idea of the BOCHICA framework (Section 3), and provides on overview of the extension interfaces (Section 4). The second part shows how to customize the framework for developing agents for virtual worlds (Section 5). Finally, Section 6 discusses the related work and Section 7 concludes this paper.

2 AGENT-ORIENTED SOFTWARE ENGINEERING

Raising the level of abstraction in software development was always an important driver in computer science research. The level of abstraction of a software language can be defined as the distance between the computer hardware and the concepts of that language (Kleppe, 2008). Since the invention of computer systems, the level of abstraction was steadily increased from opcodes, assembler languages, procedural languages, up to object oriented languages. The question that arises is how the next higher level of abstraction looks like. According to (Kleppe, 2008), "*The challenge for language engineers is that the software languages that we need to create must be at a higher level of abstraction, a level we are not yet familiar with. They must be well designed, and software developers must be able to intermix the use of multiple languages. Here too, we are facing a new, unknown, and uncertain task.*" In the agent community, AOSE has been seen as a natural successor of OOSE for a long time. Several articles discuss why AOSE has not arrived yet in mainstream software engineering (Belecheanu et al., 2006)(Jennings and Wooldridge, 2000)(McKean et al., 2008). Three of the identified main problems are (i) misunderstandings or wrong assumptions by non-agent experts, (ii) agent-oriented standards and methods are not yet sufficient for industry needs, (iii) lack of powerful tools. However, regarding the level of abstraction (Belecheanu et al., 2006) concludes: "*With concepts such as roles and responsibilities, agent-oriented approaches to problem and system description are much closer to the ways in which managers conceive of business domains than are traditional software engineering descriptions.*" This matches the requirements stated by Kleppe. Our research hypothesis is that agent technology can embody concepts like goals, roles, and organizational structures in order to build the next higher level of abstraction.

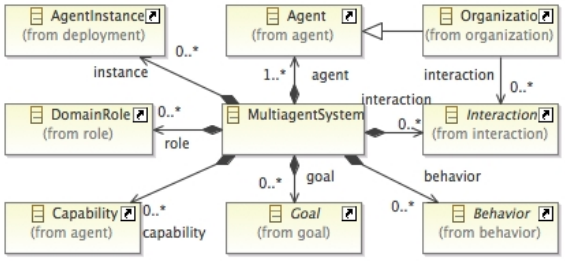

Figure 2: The central concepts of PIM4AGENTS.

The model-driven framework presented in this paper is based on the *Domain Specific Modeling Language for Multiagent Systems* (DSML4MAS) (Hahn et al., 2009). DSML4MAS is a platform independent graphical modeling language and covers the core aspects of multiagent systems (MAS), such as agents and organizations, interaction protocols, goals, behaviors, deployment aspects, etc. Its abstract syntax is defined by the *Platform Independent Metamodel for Agents* (PIM4AGENTS). The concrete syntax is specified by mappings between PIM4AGENTS concepts and graphical symbols. The semantics of the language has been formally defined in Object-Z and was manually transferred to *Object Constraint Language*[2] (OCL)-based constraints for validating PIM4AGENTS models. Model validation on a platform independent level already prevents many errors in early phases of a project. In the previous work conceptual mappings between PIM4AGENTS and the concepts of the execution platforms Jack[3] and Jade[4] were specified (Hahn et al., 2009). DSML4MAS is platform independent but it inherently possesses different *degrees of abstraction*. 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). Figure 2 depicts the core of the PIM4AGENTS metamodel.

[2]http://www.omg.org/spec/OCL/2.0/PDF/
[3]http://aosgrp.com/products/jack/index.html
[4]http://jade.tilab.com/

During the recent years, PIM4AGENTS was continuously further developed and reached a mature state. The platform, domain, and methodology independent nature of DSML4MAS makes it the perfect language for building an extensible model-driven framework on top of it.

3 FRAMEWORK OVERVIEW

Before we go into the technical details, we present the overall vision of how we think that main stream agent-based software should be developed using the BOCHICA framework. For this purpose, Section 3.1 introduces the involved stakeholders. Afterwards, Section 3.2 describes our ideas for customizing the framework for different application domains and other user demands. Section 3.3 explains how collaboration can leverage the reuse of agent-oriented design patterns. Finally, Section 3.4 describes how to incorporate results from the agent research community.

3.1 Stakeholders

The application of the BOCHICA framework requires the interplay of different stakeholders. In the following we characterize the involved parties and define their tasks.

Agent Researcher. The research area of multiagent systems is still young and many concepts are still under research. The task of the agent researcher is threefold: (i) new concepts and methods have to be developed, (ii) research results regarding properties or limitations of model elements (such as interaction protocols) have to be integrated into model repositories, and (iii) our framework needs to be grounded in a theoretical agent framework.

Language Engineer. Since BOCHICA is based on a DSL, the language engineer is responsible for extending it with new concepts. Detailed knowledge of DSML4MAS is required to align new concepts to existing ones. We distinguish between language engineers who further develop the core modeling language and those who create 3rd party plug-ins. The language engineer has to choose the right level of abstraction for the conceptual extensions.

Tool Developer. The tool developer is responsible for building the development environment based on the DSL. This includes (i) writing model transformations (ii) creating new or extended diagrams, and (iii) providing further usability extensions such as wizards and additional tools. He has to make sure that tools cover the (required) functionality of the target platform.

Agent Engineer. The agent engineer is the end user of the development environment. According to his needs, he installs the required plug-ins and uses an agent methodology to design a multiagent system for a certain scenario. He uses a model repository to cooperate with colleagues and reuses existing model artifacts. The agent engineer is also responsible to refine the generated code where necessary.

3.2 Customization

In our opinion, the assumption that one metamodel or DSL can cover the whole spectrum of agent-based applications is not realistic. The platform independent core modeling language of our framework contains generic concepts that are relevant for a wide area of agent-based applications (interaction protocols, organizational structures, behaviors, etc.). However, for many application domains it is desirable to specialize these concepts in order to improve the expressiveness of the models (e.g. for modeling agents for virtual worlds, electronic business, or agent-based simulation). It might also be the case that agent researchers want to integrate their research results by providing plug-ins that extend DSML4MAS with new concepts (e.g. commitments or new agent architectures). What we want to prevent by the extension mechanism is that the core metamodel gets cluttered by concepts that are only relevant for a small sub-set of applications. The benefit of our framework is that the mature core can be re-used in many application domains and only needs punctual extensions. The same applies to model transformations which are used to produce executable code for a certain execution environment. Existing transformations can be reused and only need to be changed for the extended aspects. Our vision is that the core of our framework will evolve over time in a community process where other researchers can contribute their ideas on how to develop multiagent systems.

3.3 Collaboration

Our vision regarding the collaboration of agent engineers is that model repositories will become available for sharing design patterns and model artifacts such as interaction protocols, organizational structures, capabilities, or behavior templates. For example, agent engineers use organizational structures and and/or decomposition trees for decomposing complex problems into sub-problems. Those patterns can be reused as blue print for similar scenarios and execution plat-

forms. One further building block is the reverse engineering approach presented in (Warwas et al., 2011) that shows how those design patterns can be extracted from already implemented multiagent systems. Experts from different agent areas can contribute new and validated artifacts to the repositories so that they become publicly available. Currently, we are using a file-based approach for sharing model artifacts but native model repositories are already becoming available (e.g. CDO[5]).

3.4 Theoretical Foundation

Wee see our framework as a bridge between agent research and software development. Metamodels are very well suited for discussing and aligning new concepts from different research areas to each other. At the same time, metamodels are the foundation for MDSD. Model transformations are used to map concepts to concrete executable artifacts. What would also be interesting is an alignment of PIM4AGENTS to theoretical agent frameworks. Research results (e.g. about the properties of a specific auction protocol) can be directly linked to model artifacts in the model repository.

4 FRAMEWORK INTERFACES

In order to customize the BOCHICA framework for certain application domains it offers various interfaces which can be extended through external plug-ins. The remainder of this section provides an overview of those interfaces. Concrete examples will be given in Section 5.

4.1 Conceptual Extension

The extension of the BOCHICA framework with new concepts is enabled by several interface concepts of DSML4MAS such as `Agent`, `Interaction`, `Resource`, `Task`, or `Service` that can be specialized by external plug-ins. The extension can be for (i) introducing new ways of modeling existing aspects (e.g. behaviors or interaction protocols), (ii) introducing completely new aspects, or (iii) specializations for a certain application domain or execution environment. The benefit of extending our framework in opposite to creating a completely new language is that large parts, which are common to most multiagent systems, can be reused. The core concepts will evolve over time and will build a solid foundation for AOSE. As an example how the BOCHICA framework could be extended is the approach for an alternative (declarative) way of modeling interaction protocols with DSML4MAS presented in (Leon-Soto, 2009). The presented approach extends the `Interaction` concepts and adds custom diagrams. At the same time, the extension is integrated into the overall framework. Users can choose which approach to apply. Technically, the extension mechanism is based on the Eclipse OSGi[6] framework and the *Eclipse Modeling Framework* (EMF) (Steinberg et al., 2008).

4.2 Information Model

The ability of creating new information models or reuse existing ones is essential for the application of an agent modeling language to real world applications. During the recent years the information model aspect of DSML4MAS evolved over several iterations and reached a sophisticated level. The DSML4MAS information model consists of four parts (see Figure 3). The core of the information model has been separated from the actual PIM4AGENTS metamodel and is based on the Ecore metamodel of the EMF framework (Steinberg et al., 2008). The Ecore metamodel is used to model classes and their attributes and relations among each other. The reuse of Ecore has several advantages: we get (i) graphical modeling support (UML class diagram style) and (ii) import from UML, XML schema (including XML de-/serialization) and existing Java code for free. Types defined in an Ecore-based information model can be made available within DSML4MAS by the concept `EType` (see Figure 3). On top of the Ecore metamodel, DSML4MAS defines basic data structures such as `Sequence`, `Set`, or `HashMap`. The third layer consists of special purpose data structures like the concept `ProtocolContext` which is used to store information for managing the execution of an interaction protocol. It is used in plans to access the current conversation context (e.g. the participants and the current state of the conversation). So far, we described the types which build the interface to the outside of the agent system. The fourth layer of the information model are internal types such as `Agent`, `Event`, or `Goal`. These internal types are required for accessing model artifacts inside a plan (e.g. the parameters of a goal). The information model of DSML4MAS can be extended by external plug-ins. One use case would be to introduce specialized data structures similar to the `ProtocolContext` (e.g. specialized result sets for querying legacy knowledge bases). Technically, the user defined data structures use the same

[5] http://www.eclipse.org/cdo/

[6] http://eclipse.org/equinox/

extension interfaces as the conceptual extensions in the previous section.

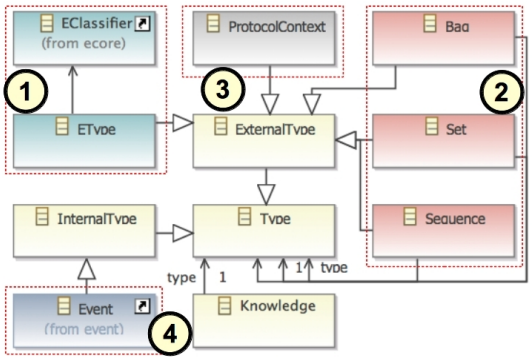

Figure 3: The four layers of the PIM4AGENTS information model: (1) basic Ecore types, (2) basic data structures, (3) specialized data structures, (4) internal types.

4.3 Language Interfaces

There exists a large number of software languages that are relevant for developing agent-based systems such as (i) knowledge representation languages (e.g. OWL), (ii) query languages (e.g. SPARQL, SQL), (iii) rule languages (e.g. SWRL, PROLOG), (iv) communication languages (e.g. KIF, FIPA ACL), programming languages (e.g. Java). A software language is always developed with a certain purpose in mind. Thus, it depends on the concrete use case which one to use. DSML4MAS provides abstract language interfaces which can be extended by external language plug-ins (see Figure 4). The main concept is `Expression`. There exist several specialized expression types such as `BooleanExpression`, `Rule`, or `ContextCondition`. The abstract expression types are used throughout DSML4MAS. For example, an `AchieveGoal` has a target and failure condition of type `BooleanExpression` and a `Plan` has a context condition of type `ContextCondition`. External plug-ins can specialize the abstract expression types with concrete languages. We assume that an external language is also based on Ecore. This is not a hard restriction since more and more software languages, such as SPARQL or Java, are becoming available in public metamodel zoos (e.g. EMFText concrete syntax zoo[7], Atlantic metamodel zoo[8]). We use a reflection-based approach for parsing user defined expression strings into a language specific expression model (interface concept `EObject`) and assign it to the `Expression` object's `object` attribute. The ability to plug in 3rd party languages to BOCHICA makes the approach very flexible. For example, an external semantic web plug-in could extend (i) the `KnowledgeBase` concept for an OWL-based knowledge base, (ii) the `Expression` concept with SPARQL for reasoning about an agent's beliefs, and (iii) custom result sets for SPARQL queries (as explained in the last section). The benefit of our approach is that technical details, such as the integration of the knowledge base and SPARQL into the concrete agent execution platform, are hidden on the modeling level. At the same time, models can be tailored to certain application domains. Of course, the integration at the platform level has to be done at some point (we discuss it later) but the agent engineer can concentrate on the design of the overall system.

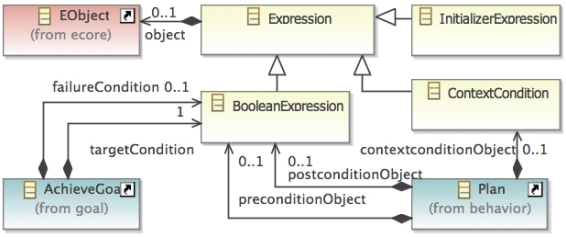

Figure 4: Expression interface of PIM4AGENTS.

4.4 Methodologies

During the recent years, several agent-oriented methodologies have been proposed. Most of the developed approaches are based on a graphical modeling language (see discussion in Section 6). The focus of our approach was always on developing an expressive platform independent agent modeling language that can be used for model-driven development of agent-based systems and less on the methodology part. Since both aspects are complementary, our idea is to use methodology plug-ins for extending the BOCHICA framework. In the same way as BOCHICA can be extended with new agent concepts, methodology providers can contribute plug-ins with new views and methodology concepts. For example, the Prometheus methodology (Padgham et al., 2004) collects in the *system specification* phase abstract functionalities and goals of a SUC. In the *architectural design* phase the functionalities and goals are grouped to agents. A Prometheus plug-in for BOCHICA could extend the framework with the missing concepts for collecting abstract functionalities in the system specification phase (since it is not covered by DSML4MAS). The architectural design phase could be based on existing concepts of DSML4MAS. As interface, DSML4MAS provides the concept `MethodologyArtifact`. The methodology

[7] http://www.emftext.org/

[8] http://www.emn.fr/z-info/atlanmod/index.php/Atlantic

aspect is not part of this paper. Instead of having a separate modeling language and tool for each methodology, most of the methodologies could be integrated into one framework and share a common core. This would join the efforts of the involved parties and would ease the maintenance of the tool chain.

4.5 Custom Views and Tools

Views are used in graphical modeling languages to visualize the relations of model artifacts of a certain sub-aspect of a complex model. DSML4MAS provides views for modeling agent types and organizational structures, behavior templates, interaction protocols, etc. 3rd party developers can use the extension interface to provide own views. Especially, when DSML4MAS is customized for a certain application domain, new views can help to adapt it to the user's needs. Technically, diagrams and tools can be plugged into the framework by using the extension point mechanism provided by the Eclipse OSGi framework and GMF[9].

4.6 Transformations

Model transformations in model-driven development are used to gradually refine a model of a SUC until executable code is generated. We assume that there exist base transformations from DSML4MAS to agent execution platforms such as Jack or Jadex[10]. A base transformation maps the concepts of DSML4MAS to executable artifacts of the agent platform. The transformation uses design patterns to produce clean code which can be manually refined where needed. As DSML4MAS gets extended with new concepts, an existing base transformation is no longer complete regarding the covered concepts. Thus, the affected model transformations also have to be extended in order to reflect the changes applied to the metamodels. We see three possibilities how this can be achieved. Some model transformation languages (e.g. QVT[11]) allow to write a new transformation which inherits from an existing one. Thus, an existing mapping rule can be overloaded by a new and extended one. Other transformation languages like XPAND[12] use an aspect-orientated approach for hooking into an existing transformation and extending it. A further possibility is to chain transformations, where the first one is a base transformation and the succeeding one "patches" the result of the proceeding one. An external plug-in usually exists of a conceptual extension to DSML4MAS and the according (updated) model transformations. Reusing existing model transformations reduces development costs and time and increases code quality by using well established design patterns.

5 VIRTUAL REALITY EXTENSION

Today, intelligent behavior of avatars in virtual worlds is usually simulated by triggered script sequences which create the illusion of intelligent behavior for the user. However, the flexibility of those avatars is, due to their static nature, very limited. In the research project *Intelligent Simulated Realities* (ISReal) our research group developed the first simulation platform based on semantic web technology for bringing intelligent behavior into virtual worlds (Kapahnke et al., 2010). The basic idea of ISReal was to use semantic web technology to extend purely geometric objects with ontological information (OWL[13]-based; e.g. concept door links two rooms and can be open or closed) and specify their functionality by semantic service descriptions (OWL-S[14] based; e.g. open and close door service), called *object services*. Intelligent agents perceive this information, store it in their knowledge base, and use it for reasoning and planning. An object service is grounded in a service implementation which invokes according animations or simulation modules. The platform consists of various simulation components which can be distributed in a network. Before we show how we extended our model-driven framework for developing ISReal agents, we introduce the main components of the ISReal platform.

Global Semantic Environment. The *Global Semantic Environment* (GSE) maintains the global ontological facts of the virtual world. It is responsible for (i) executing object services (e.g. checking the precondition and invoking the service grounding), (ii) updating facts (e.g. when a door gets closed), and (iii) handling queries (e.g. SPARQL).

Agent Environment. The ISReal agent environment defines interfaces for connecting 3rd party agent execution platforms to the ISReal platform (we currently use Jack, Jadex, and the Xaitment[15] game AI engine). Every ISReal agent is equipped with a *Lo-*

[9] www.eclipse.org/gmf
[10] http://jadex.sourceforge.net
[11] http://www.omg.org/spec/QVT/1.0/
[12] http://www.eclipse.org/modeling/m2t/
[13] http://www.w3.org/TR/owl-primer/
[14] http://www.w3.org/Submission/OWL-S/
[15] http://www.xaitment.com/

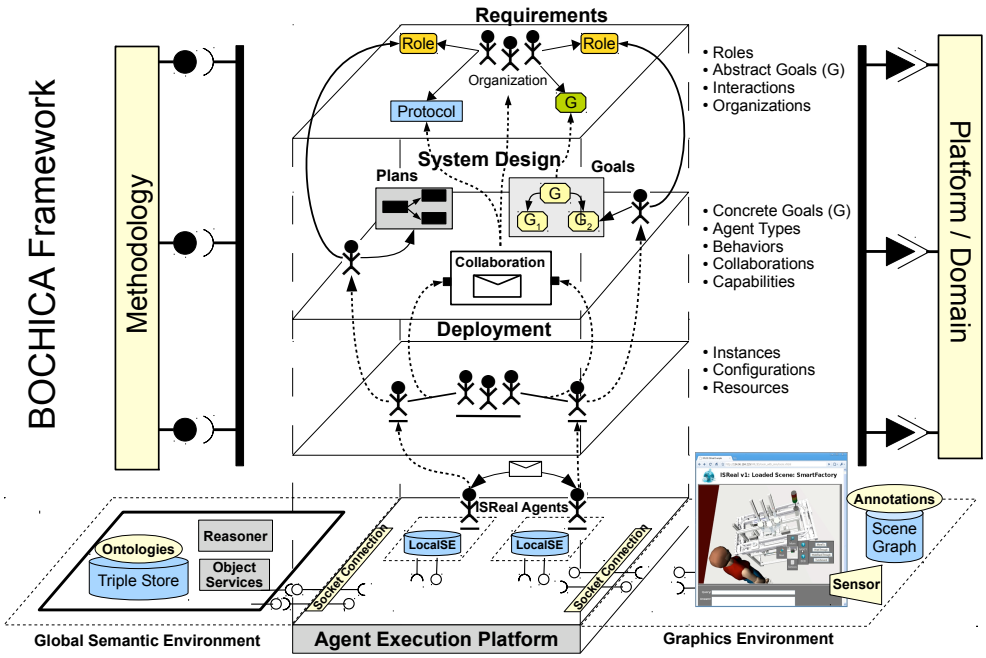

Figure 5: The bottom layer depicts the components of the ISReal platform. The upper central part shows the inherent degrees of abstraction of DSML4MAS. The left an right hand side represent the interfaces for extending DSML4MAS.

cal Semantic Environment (LSE) which is an agent's local knowledge base. The LSE stores the perceived information and enables the agent to reason about it. Moreover, the LSE is equipped with an AI planner.

Graphics Environment. As user interface to the platform we use an XML3D[16]-enabled standard web browser. The 3D scene graph is part of the browser's *Document Object Model* (DOM) and can be manipulated using Java Script. It also contains RDFa[17]-based semantic annotations of the 3D-objects (concept URI, object URI, and semantic service URIs). Moreover, we extended the browser with an agent perception component which allows agents to sense the annotated 3D objects.

An intelligent ISReal avatar consists of (i) the geometrical shape (body) and animations, (ii) a perception component, (iii) semantic annotations, and (iv) an agent that processes the perceived information and controls the body. Our idea is now to customize the BOCHICA framework in order to get a specialized development environment for constructing ISReal agents. Artifacts such as the geometrical shape or ontologies are developed in specialized (external) tools. DSML4MAS has to be extended with interface concepts for those artifacts in order to create a complete ISReal agent specifications. Figure 5 depicts the big picture of how we think that intelligent agents for the ISReal platform should be developed. For a detailed introduction to the ISReal platform we refer to (Kapahnke et al., 2010). The remainder of this section discusses the customizations of the BOCHICA framework.

[16]http://www.xml3d.org

[17]http://www.w3.org/TR/xhtml-rdfa-primer/

5.1 CONCEPTUAL EXTENSION

Figure 6 depicts some of the extensions of the PIM4AGENTS metamodel for ISReal. The upper row shows core concepts of PIM4AGENTS. The `OMSConfig` concept is the root of a metamodel which is used in the ISReal platform for configuring the LSE with concrete ontologies, object services, etc. The imported `OMSConfig` concept of the ISReal platform is reused by the extension plug-in. The middle row de-

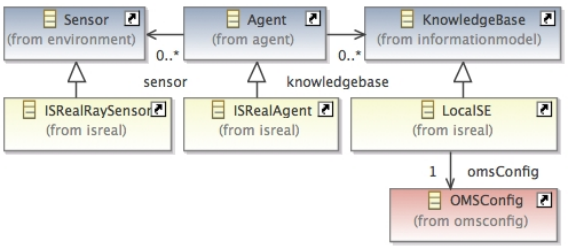

Figure 6: A part of the ISReal extension of PIM4AGENTS. The upper row shows native concepts of DSML4MAS. The middle row depicts specialized ISReal concepts. The bottom row shows model artifacts of the ISReal platform.

picts the actual extension of the PIM4AGENTS metamodel. The `ISRealAgent` is a PIM4AGENTS agent which has a URI that identifies an agent's ontological type, an `ISRealRaySensor` (resolution, update rate), a LSE, and a (not visualized) link to a graphical avatar (the agent's body). Some concepts of PIM4AGENTS change their technical meaning when they are transformed to the ISReal platform. For example, ISReal agents use their plans to orchestrate object services. PIM4AGENTS already provides support for orchestrating traditional web services by plans. Since ISReal object services are very similar to ordinary web services, the existing concepts can be reused without modification. Figure 7 depicts a very simple plan that is triggered by the `MoveToGoal` and invokes the `MoveNearService` with the according parameters. The `MoveNearService` is used for in-room navigation (no path finding across rooms). The plan's context condition checks whether the target object is located in the same room. The ISReal model transformation will generate code for invoking an ISReal object service instead of an ordinary web service class (see Section 5.3).

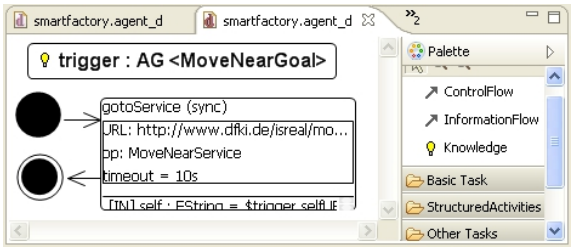

Figure 7: Agent-based orchestration of ISReal object services (behavior diagram).

5.2 LANGUAGE EXTENSION

In order to make rational decisions, it is essential for agents to reason about the perceived information. The interface to a knowledge base is usually defined by a query language. As ISReal agents are based on semantic web technology, we decided to use SPARQL queries to access the LSE. Two of the application scenarios are (i) to use SPARQL-Ask queries to define the target condition of achieve goals and (ii) to specify the context condition of plans with SPARQL-Select queries. As explained in Section 4.3, PIM4AGENTS defines language interface concepts such as `BooleanExpression` that are used throughout the metamodel. We reused an Ecore-based SPARQL DSL which is provided by the EMFText concrete syntax zoo to extend DSML4MAS. The `BooleanExpression` was extended with SPARQL-Ask and the `ContextCondition` with SPARQL-Select. We also reused the automatically generated parser of EMFText for parsing SPARQL text queries into SPARQL models that are plugged into the DSML4MAS extension slot. Figure 8 depicts an example PIM4AGENTS `AchieveGoal` for walking to an object. The target condition checks the predicate `nearAt(self, object)` which is computed by the graphics environment and the GSE.

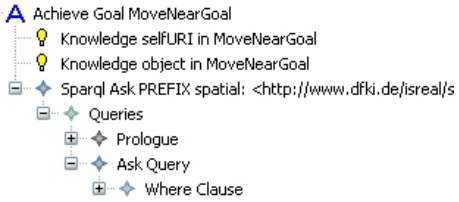

Figure 8: SPARQL target condition in PIM4AGENTS.

5.3 TRANSFORMATIONS

In Section 4.6, we explained that we expect to have base transformations for mapping PIM4AGENTS models to different execution platforms. An existing base transformation has to be extended as BOCHICA gets extended for a new application domain (and the target platform is supposed to support the extension). Now, we explain how we extended the base transformation to the Jadex platform. A Jadex application consists of XML-based configuration files for applications, agents, and capabilities. Behaviors are encoded in Java-based plans. The base transformation from DSML4MAS to Jadex consists of the four modules Application, Agent, Capability, and Behavior (see Figure 9). The first three modules map concepts from PIM4AGENTS to the Jadex metamodel (green arrows) using QVT model-to-model transformations. The generated Jadex model is automatically serialized to valid Jadex XML files by EMF. We decided not to create a separate Jadex metamodel for plans. This decision was made due to experiences with previous transformations to Jack and Jade (to avoid overhead and simplify extensions). The model-to-text transformation is done using XPAND. The separation into separate modules leverages extensibility and eases maintenance. The information model is based on Ecore and we rely on the capability of EMF to automatically serialize types to Java code (white arrow). Since we want to focus on the overall approach, the details of the transformations and the platform specific metamodel for Jadex will not be discussed in this paper.

The blue parts in Figure 9 depict (i) the ISReal extension to the PIM4AGENTS metamodel, (ii) the ISReal extension to the Jadex QVT and XPand base

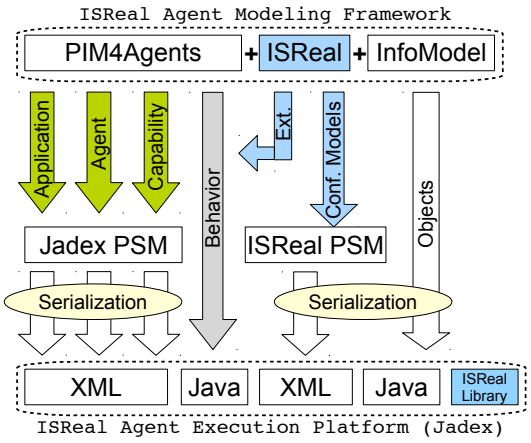

Figure 9: ISReal transformation overview.

transformations, (iii) the generation of ISReal configuration files for the ISReal platform, and (iv) an additional ISReal library that enables Jadex for the ISReal platform. The ISReal library implements the interfaces of the ISReal agent environment for passing incoming perception events and user queries to the agents running in the Jadex platform. Moreover, it includes Jadex into the start-up procedure of the distributed ISReal platform and provides an ISReal capability which makes an Jadex agent to an ISReal agent. For example, it provides access to the LSE. Now, we provide an example on how the invocation of a web service in PIM4AGENTS can be adapted by a transformation extension to invoke object services. Figure 10 depicts a part of the XPAND-based transformation which uses aspect-orientation for replacing the `InvokeWS2ServiceInvokation` rule for generating the web service invocation code with the invocation of an ISReal object service. The first part sets the variable bindings of the object service and the second part does the actual invocation through a helper class from the ISReal library.

5.4 ISREAL VIEW

The technical details explained so far are (in the ideal case) not visible to the end user. He is guided by a methodology and uses graphical diagrams to design a multiagent system for a certain use case. The graphical front end abstracts from technical details such as (i) the integration of Jadex into the ISReal platform, (ii) the invocation of ISReal object services in Jadex, or (iii) the evaluation of SPARQL queries in the LSE. Custom views can be created to show new aspects or show existing ones in a different context. Figure 11 depicts the ISReal agent diagram which contains, in addition to the normal PIM4AGENTS artifacts, the ISReal sensor and the LSE.

```
«AROUND *InvokeWS2ServiceInvokation FOR pim4agents::task::InvokeWS»
  BindingList input«this.name.toFirstUpper()» = new BindingListImpl();
  «FOREACH this.incomingParameters AS i-»
    input«this.name.toFirstUpper()».addPair(
      "«this.serviceEndPoint»#«i.name»",
      (String)this.getBeliefbase().getBelief(
        "«getKnowledgeName(this.getContainingActivities().first())»
          _«i.value.subString(1,i.value.length)»"
      ).getFact()
    );
  «ENDFOREACH»

  Util.executeObjectService(
   (ISRealAgent)this.getBeliefbase().getBelief("isrealAgent").getFact(),
   "«this.serviceEndPoint»#«this.operationName»",
   input«this.name.toFirstUpper()»,
   «this.timeout.toInteger()*1000»);
«ENDAROUND»
```

Figure 10: This aspect-oriented mapping rule replaces the original web service invocation by an ISReal object invocation.

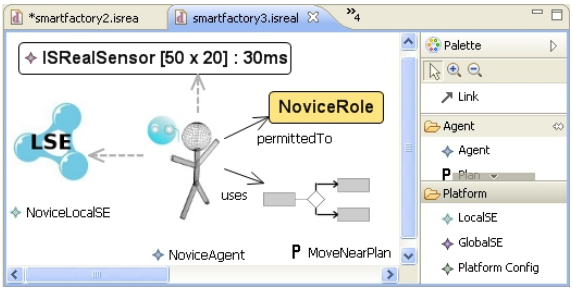

Figure 11: The customized ISReal agent diagram.

6 RELATED WORK

During the recent years, several languages for modeling agent-oriented systems have been proposed (Henderson-Sellers and Giorgini, 2005)(Sterling and Taveter, 2009). Most of the modeling languages were developed to support a certain agent methodology and do not have an own name. Their existence is bound to the methodology. Unfortunately, the majority of the developed modeling tools are only partially based on standardized technology for model-driven development[18] which hampers the benefits of MDSD. For example, the *Prometheus Design Tool*[19] (Prometheus methodology) has no explicit underlying metamodel. AgentTool III[20] (O-MaSE), INGENIAS Development Kit[21] (INGENIAS), Taom4e[22] (Tropos), and REBEL[23] (ROADMAP) are based on Ecore metamodels but use legacy or non-MDA-based model transformations. To the best of our knowledge, the mentioned approaches are not thought to be extended or customized by 3rd party plug-ins. Beside the methodology-based modeling languages, there exist also approaches for extending the *Unified Modeling Language* (UML) with agent concepts (e.g. Object Management Group's (OMG) *Agent Meta-*

[18] We analyzed the publicly available software.
[19] http://www.cs.rmit.edu.au/agents/pdt/
[20] http://agenttool.cis.ksu.edu/
[21] http://ingenias.sourceforge.net/
[22] http://selab.fbk.eu/taom/
[23] http://www.agentlab.unimelb.edu.au/software.html

model and Profile[24] (AMP) or FIPA Agent UML[25]). Those approaches promise to reuse the ecosystem built around UML – including the large user group. However, modeling agents is fundamentally different from objects. Agents possess an internal mental model and require different methods and design patterns. Moreover, our experiences in AMP showed that it is hard to extend UML since the underlying *Meta Object Facility* (MOF) metamodel is complex and extensions of existing elements have many not desired and non-obvious implications. Thus, we are sceptical that extending UML in its current form suffices the needs of AOSE. UML, which is a general purpose modeling language, offers two extension mechanisms: (i) heavy weight metamodel extensions and (ii) light weight profiles. Metamodel extensions of UML underlie a standardization process of OMG and are not for the normal end user. Profile-based extensions can be created by end users and allow a limited customization. An alternative to our approach would be the creation of a platform specific modeling language (e.g. for the ISReal-enabled Jadex platform). In (Kardas et al., 2009) two platform specific modeling languages for the agent platforms SEAGENT (Dikenelli, 2008) and Jadex were presented. The possibility to customize the language if the agent platform (e.g. Jadex) is integrated into a larger platform is not provided. Moreover, it is unclear how Java-based plans are modeled for the Jadex.

7 CONCLUSIONS

In this paper we presented a novel model-driven framework for AOSE which integrates the experiences we gained during the recent years with modeling MAS. The BOCHICA framework goes beyond the state of the art in AOSE as it is not created for a certain execution platform, methodology, or application domain. Instead, it is based on a platform independent agent core modeling language and provides generic extension interfaces for integrating results from agent research as well as for customizing it regarding user-specific application domains and platforms. After we presented our vision on how to apply our framework in Section 3, the extension interfaces were introduced in Section 4. We evaluated the framework on a semantic web based distributed simulated reality platform. Our approach is especially suited for large scale applications or target environments with many end-users (e.g. the ISReal platform) where customizations pay off. Small applications can be realized with the functionality provided by the core modeling language and do not need customizations. We see our approach as a contribution to the unification of the diverse field of agent-oriented modeling and to bridge research and software development.

[24]http://www.omg.org/cgi-bin/doc?ad/08-09-05.pdf

[25]http://www.auml.org/

REFERENCES

Belecheanu, R. A. et al. (2006). Commercial applications of agents: Lessons, experiences and challenges. *5th Int. joint Conf. on Autonomous Agents and Multi-Agent Systems.*, pages 1549—1555.

Dikenelli, O. (2008). SEAGENT MAS platform development environment. In *Proc. of the 7th Int. joint Conf. on Autonomous agents and multiagent systems: demo papers*, AAMAS '08, pages 1671–1672. IFAAMAS.

Hahn, C. et al. (2009). A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18:239–266.

Henderson-Sellers, B. and Giorgini, P. (2005). *Agent-Oriented Methodologies*. Igi Global.

Jennings, N. R. and Wooldridge, M. (2000). Agent-Oriented Software Engineering. *Artificial Intelligence*, 117:277—296.

Kapahnke, P. et al. (2010). ISReal: an open platform for semantic-based 3D simulations in the 3D internet. In *Proc. of the 9th Int. Semantic Web Conference on the Semantic Web (ISWC'10)*, page 161–176. Springer.

Kardas, G., Ekinci, E. E., Afsar, B., Dikenelli, O., and Topaloglu, N. Y. (2009). Modeling tools for platform specific design of Multi-Agent systems. In *Proc. of the 8th German Conf. on Multiagent System Technologies (MATES'10)*, volume 5774, pages 202–207. Springer.

Kleppe, A. (2008). *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Longman, Amsterdam, 1 edition.

Leon-Soto, E. (2009). Modelling interaction protocols as modular and reusable 1st class objects. In *Agent-Based Technologies and Applications for Enterprise Interoperability*, volume 25 of *LNBIP*, pages 174–219. Springer.

McKean, J. et al. (2008). Technology diffusion: analysing the diffusion of agent technologies. *Autonomous Agents and Multi-Agent Systems*, 17(3):372–396.

Padgham, L. et al. (2004). *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & Sons.

Steinberg, D. et al. (2008). *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2nd revised edition.

Sterling, L. and Taveter, K. (2009). *The Art of Agent-Oriented Modeling*. The MIT Press.

Warwas, S. et al. (2011). Making multiagent system designs reusable: A model-driven approach (accepted). IEEE Computer Society Press.