BOCHICA: A MODEL-DRIVEN FRAMEWORK FOR ENGINEERING MULTIAGENT SYSTEMS

Stefan Warwas, Klaus Fischer, Matthias Klusch and Philipp Slusallek
German Research Center for Artificial Intelligence (DFKI), Campus D3.2, 66123, Saarbrücken, Germany

Keywords: Agent-oriented software engineering, Modeling framework.

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 virtual reality 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 similar scenarios. Current agent modeling languages cover the core concepts of multiagent systems but are neither thought to be customized for a certain application domain nor to be extended by external researchers with new or alternative AI and agent concepts. In this paper we propose a model-driven framework for engineering multiagent systems, called BOCHICA, which is based on a platform independent core modeling language and can be tailored through several extension interfaces to the user's needs. The framework leverages the reuse of existing design patterns and reduces development time and costs for creating application domain specific modeling solutions. We evaluated our approach on a distributed semantic web based execution platform for virtual worlds.

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* (AI) 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 until concrete code can be generated. Several core aspects of MDSD were standardized by the *Object Management Group* (OMG) as *Model-driven Architecture* (MDA).

During the recent years, several approaches for modeling agent-based systems have been proposed (Sterling and Taveter, 2009). 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. An ideal modeling language would contain, beside the core concepts of *Multiagent Systems* (MAS), specific concepts, software languages, graphical representations, etc. for a certain target environment. Moreover, it is desirable to extend the modeling language with new concepts from AI and agent research (e.g. new ways of modeling behaviors). In this paper we propose a model-driven framework for engineering agent-based systems to overcome the mentioned limitations. Our framework, called BOCHICA[1], is based on a core *Domain Specific Language* (DSL) which covers the main aspects

[1]Bochica was a semi-god of the Muisca culture who brought them living skills and showed them how to organize their lives.

of MAS (see Figure 1). The framework provides several interfaces for new concepts, methods, and 3rd party software languages. We envision BOCHICA as a bridge between agent research and 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.

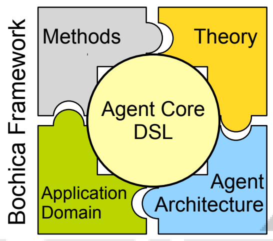


Figure 1: The BOCHICA framework for AOSE is based on a platform independent core DSL and is able to integrate research results, methods, and application domains.

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 creating an application domain specific agent development environment for semantic virtual worlds (Section 5). Finally, Section 6 discusses the related work and Section 7 concludes this paper.

2 BACKGROUND

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 modeling languages of the next higher level of abstraction.

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 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). *Object Constraint Language*[2] (OCL)-based constraints are used for validating PIM4AGENTS models. Model validation on a platform independent level already prevents many errors in early phases of a project. 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). The platform, domain, and methodology independent nature of DSML4MAS makes it the perfect language for building an extensible framework around it. For this purpose, we introduce the overall idea behind BOCHICA and present the made extensions.

3 FRAMEWORK OVERVIEW

Before we go into the technical details, we present the overall vision of how we think that large scale agent-based software should be developed using the

[2] http://www.omg.org/spec/OCL/2.0/PDF/

BOCHICA framework.

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 MAS 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 to bridge research and software development.

Language Engineer. Since BOCHICA is based on a DSL, the language engineer is responsible for extending it with new concepts. Detailed knowledge of the core DSL 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 BOCHICA. 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 the tools and transformations 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 MAS 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 BOCHICA with new concepts (e.g. commitments or new agent architectures). What we want to prevent by the extension mechanism is that the core DSL gets cluttered by concepts that are only relevant for a small sub-set of applications. The benefit of BOCHICA is that the mature core can be reused 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 extended for the changed 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 MAS.

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 platforms. In (Warwas et al., 2011), an approach for making the underlying design of concrete implemented MAS was presented. Agent experts can use this approach for sharing new and validated artifacts to the repositories which proved their use in practical applications. Currently, we are using a file-based approach for sharing model artifacts but native model repositories are already becoming available (e.g. CDO[3]).

3.4 Theoretical Foundation

Wee see BOCHICA as a bridge between agent research and concrete 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. What would also be interesting is an alignment of BOCHICA to theoretical agent frameworks. Research results (e.g. about the properties of a specific auction protocol) can be directly linked to model

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

artifacts in the model repository. This information helps engineers in constructing MAS.

4 FRAMEWORK INTERFACES

In order to customize the BOCHICA framework it offers various interfaces which can be extended through external Eclipse-based plug-ins. The remainder of this section provides an overview of those interfaces. Examples will be given in Section 5.

4.1 Conceptual Extension

BOCHICA can be extended with new concepts for (i) introducing new ways of modeling existing aspects (e.g. behaviors), (ii) introducing completely new aspects (e.g. commitments), or (iii) specializations for a certain application domain or execution environment. The extension is enabled by several interface concepts such as `Agent`, `Interaction`, `Resource`, or `Task` that can be specialized by external plug-ins. The benefit of extending our framework in opposite to creating a completely new approach is that large parts, which are common to most MAS, can be reused. The core concepts will evolve over time and will build a solid foundation for AOSE. As an example of how the BOCHICA framework can be extended is the approach for an alternative (declarative) way of modeling interaction protocols with DSML4MAS presented in (Leon-Soto, 2009). The presented approach extended the `Interaction` concept of DSML4MAS and added custom diagrams. At the time of the creation of the extension, BOCHICA was not available so that DSML4MAS had to be extended directly. Now, BOCHICA provides interfaces for 3rd party developers for extending it with new concepts without touching the core. At the same time, the extension is integrated into the overall framework. End users can choose which alternative to apply. Technically, the extension mechanism is based on the Eclipse OSGi[4] framework and the *Eclipse Modeling Framework* (EMF) (Steinberg et al., 2008).

4.2 Information Model

The information model interface of BOCHICA consists of four parts (see Figure 2). The core of the information model has been separated from BOCHICA and is based on the Ecore metamodel provided by EMF (Steinberg et al., 2008). Ecore 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 BOCHICA by the concept `EType` (see Figure 2). On top of the Ecore metamodel, BOCHICA 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 BOCHICA 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 extension interfaces as in Section 4.1.

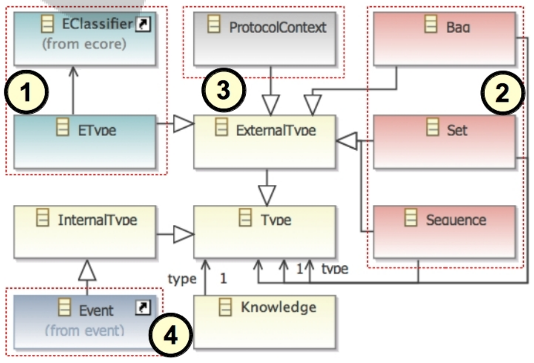


Figure 2: The four layers of the BOCHICA information model interface: (1) basic Ecore types, (2) basic data structures, (3) specialized data structures, (4) internal types.

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), (iii) rule languages (e.g. SWRL, PROLOG), (iv) communication languages (e.g. KIF, FIPA ACL), (v) 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

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

which one to use. BOCHICA provides abstract language interfaces which can be extended by external language plug-ins (see Figure 3). The main concept is `Expression`. There exist several specialized expression types such as `BooleanExpression`, `Rule`, or `ContextCondition`. The abstract expression types are used throughout the framework. 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[5], Atlantic metamodel zoo[6]). 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 (see Figure 3). This approach can be used (i) for checking the syntactical correctness of an expression, (ii) for checking whether variable symbols inside the language model are bound in the surrounding scope, and (iii) to process the expression models in model transformations. 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 a certain target environment. Of course, the integration at the platform level has to be done at some point (we discuss it later) but the agent engineer has a consistent view and can concentrate on the design of the overall system.

4.4 Methodologies

During the recent years, several agent-oriented methodologies have been proposed (Sterling and Taveter, 2009). Most of the developed approaches are supported by 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 it with the missing concepts for collecting abstract functionalities in the system specification phase (since it is not covered by the core DSL). The architectural design phase could be based on existing concepts of the core DSL. As interface, BOCHICA provides the concept `MethodologyArtifact`. 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.

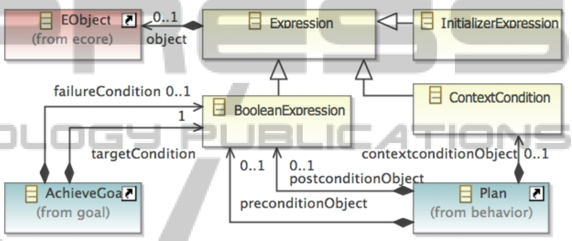


Figure 3: Expression interface of BOCHICA.

4.5 Transformations

Model transformations in MDSD are used to gradually refine a model of a SUC until executable code is generated. We assume that there exist *base transformations* from BOCHICA to agent execution platforms such as Jack or Jadex. A base transformation maps the concepts of the core DSL to executable artifacts of an agent platform. As BOCHICA gets extended with new concepts, an existing base transformation is no longer complete regarding the covered concepts. Thus, an *extension transformation* is required which extends an existing base transformation for the new concepts (if the target platform shall be enabled for the extension). We see three possibilities how this can be achieved. Some model transformation languages (e.g. QVT[7]) 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[8] 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 sup-

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

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

[7]http://www.omg.org/spec/QVT/1.0/

[8]http://www.eclipse.org/modeling/m2t/

plements the result of the proceeding one. Thus, an external plug-in for BOCHICA usually exists of (i) conceptual extensions and (ii) an extension transformation for the required target environment (assuming that the base transformation already exists). Maintaining the tool chain is one of the main problems in MDSD. Reusing existing model transformations reduces development costs and time and increases code quality by using well established design patterns.

4.6 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 system. BOCHICA provides standard views for agent types and organizational structures, protocols, goal hierarchies, deployment configurations, etc. 3rd party developers can use the extension interface for customizing BOCHICA to a certain application domain or introduce new ways of viewing existing aspects. Views can also help to adapt the development environment to certain user groups. Technically, diagrams and tools can be plugged into the framework by using the extension point mechanism of the Eclipse OSGi framework and GMF[9].

5 VIRTUAL REALITY DEVELOPMENT ENVIRONMENT

After we introduced the overall framework, we now want to show how to apply BOCHICA to a large scale real world scenario. 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-based; e.g. concept door links two rooms and can be open or closed) and specify their functionality by semantic service descriptions (OWL-S-based; e.g. open and close door services), 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 the BOCHICA framework for developing ISReal agents on the Jadex BDI platform, we introduce the main components of the ISReal platform.

[9]http://www.eclipse.org/gmf

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[10] game AI engine). Every ISReal agent is equipped with a *Local 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. The user interface of the ISReal platform is realized by an XML3D[11]-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[12]-based semantic annotations of the 3D-objects such as the concept URI, the object URI, and the semantic object service URIs. Moreover, we extended the browser with an agent sensor which allows agents to perceive 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. Artifacts such as the geometrical shape, animations, or ontologies are developed using specialized 3rd party tools. We decided to base the development environment for ISReal agents on BOCHICA and use Jadex as the target agent platform. This has several advantages: (i) BOCHICA already provides the core concepts, diagrams, etc. for modeling agent systems, (ii) we can reuse the existing base transformation to Jadex, (iii) we only need to customize the missing aspects of BOCHICA for creating an individual development environment for agents in semantic virtual worlds, and (iv) it enables the reuse of existing model artifacts (e.g. interaction protocols). Figure 4 depicts the big picture of how we think that

[10]http://www.xaitment.com/
[11]http://www.xml3d.org
[12]http://www.w3.org/TR/xhtml-rdfa-primer/

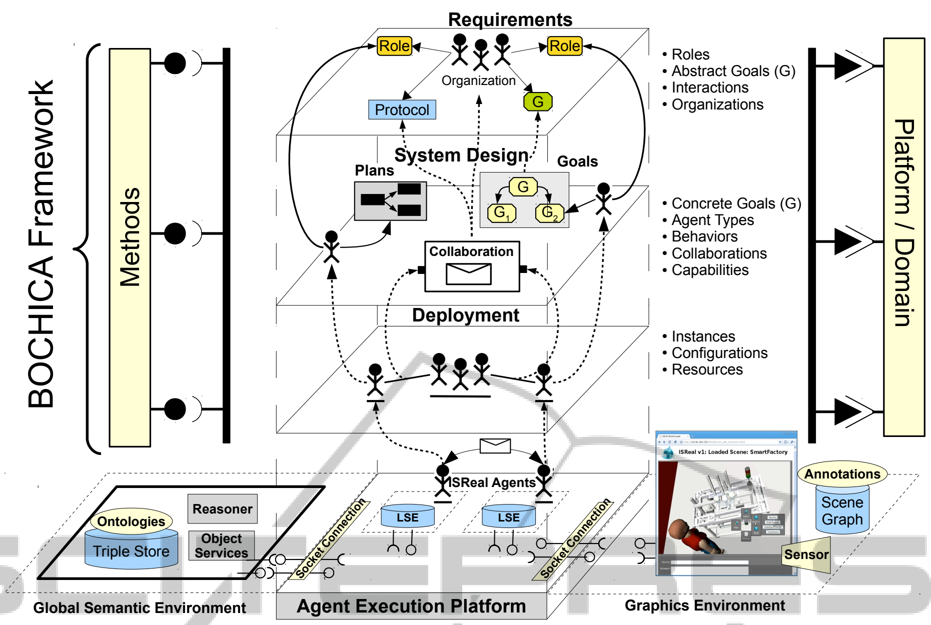


Figure 4: The bottom layer depicts the components of the ISReal platform. The upper central part shows the inherent degrees of abstraction of BOCHICA. The left an right hand side represent the interfaces for extending the framework.

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 extensions of the BOCHICA framework for developing ISReal agents.

5.1 Conceptual Extension

Figure 5 depicts some of the conceptual extensions for ISReal. The upper row shows interface concepts of BOCHICA. 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 depicts the actual conceptual extensions. The `ISRealAgent` specializes the concept `Agent` and has an URI which defines an agent's ontological type, an `ISRealRaySensor` (resolution, update rate), a LSE, and a (not visualized) reference to an existing graphical avatar (the agent's body). Some concepts of BOCHICA change their technical meaning when they are transformed to the ISReal platform. For example, ISReal agents use their plans to orchestrate object services. BOCHICA already provides support for orchestrating web services by plans. Since ISReal object services are very similar to web services, the existing concepts can be reused without modification. Figure 6 depicts a very simple plan that is triggered by the `MoveNearGoal` 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 generates code for invoking an ISReal object service instead of code for invoking an ordinary web service (see Section 5.3).

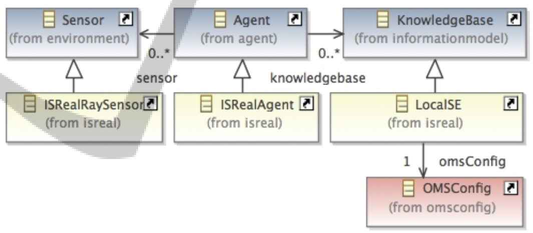
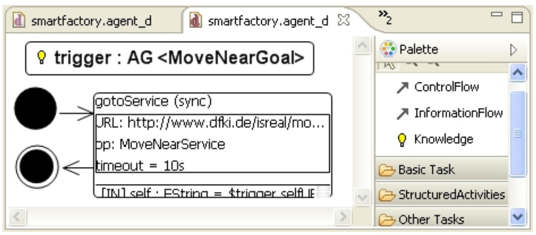


Figure 5: A part of the ISReal extension of BOCHICA.



trigger : AG <MoveNearGoal>

gotoService (sync)
URL: http://www.dfki.de/isreal/mo...
op: MoveNearService
timeout = 10s
[IN] self : EString = $trigger.self.IE

Figure 6: 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, BOCHICA defines language interface concepts such as `BooleanExpression` that are used throughout the framework. We reused an Ecore-based SPARQL DSL which is provided by the EMFText concrete syntax zoo to extend BOCHICA (see Section 4.3). 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 BOCHICA extension slot. Figure 7 depicts an example `AchieveGoal` for walking to an object. The target condition `nearAt(self, object)` is validated in the agent's LSE. The predicate is perceived by the agent through its sensor after it has been computed by the graphics environment and the GSE.

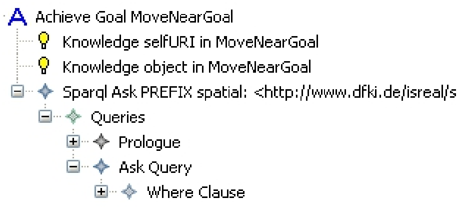

Achieve Goal MoveNearGoal
Knowledge selfURI in MoveNearGoal
Knowledge object in MoveNearGoal
Sparql Ask PREFIX spatial: <http://www.dfki.de/isreal/s
Queries
Prologue
Ask Query
Where Clause

Figure 7: `AchieveGoal` with SPARQL target condition.

5.3 Transformations

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 BOCHICA to Jadex consists of the four modules Application, Agent, Capability, and Behavior (see Figure 8). The first three modules map concepts from BOCHICA to the Jadex *Platform Specific Metamodel* (PSM) (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 of the transformation into separate modules leverages extensibility and eases maintenance. As explained in Section

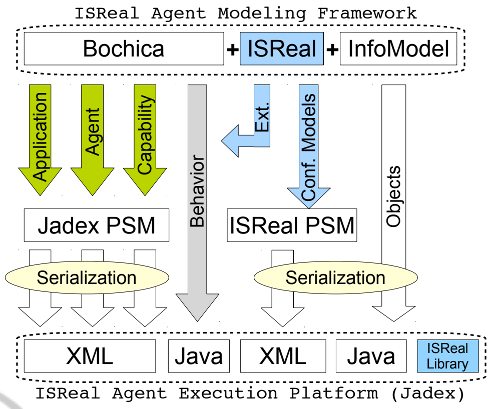


Figure 8: ISReal (Jadex) transformation overview.

4.5, 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 Jadex PSM will not be discussed in this paper. The blue parts in Figure 8 depict (i) the ISReal extension of BOCHICA, (ii) the ISReal extension to the Jadex QVT and XPand base transformations, (iii) the generation of configuration files for the ISReal agent component, 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 a Jadex agent to an ISReal agent. For example, it equips an agent with a LSE. Figure 9 depicts a XPand-based aspect-oriented mapping rule which replaces the original mapping rule of the Jadex base transformation for invoking a standard web service by 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 provided by 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 MAS 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. Figure 10 depicts the ISReal agent diagram which contains, in addition to the standard BOCHICA agent diagram

```
«AROUND template::invokews2code FOR pim4agents::InvokeWS»
  wsUri = "«this.serviceEndPoint»#«this.operationName»";
  wsBindings= new BindingListImpl();
  «FOREACH this.incomingParameters AS i-»
     wsBindings.addPair("«this.serviceEndPoint»#«i.name»",
     «resolveKnowledgeValueExpression(i.value.trim(), i, this)»);
  «ENDFOREACH-»

  OSInvocation w = new OSInvocation((ISRealAgent) this
    .getBeliefbase().getBelief("isrealAgent").getFact(), wsUri,
     wsBindings, (long) «this.timeout * 1000»);
  this.waitForExternalCondition(w);
  wsRes = w.getResult();
  if(!wsRes) this.fail();
«ENDAROUND»
```

Figure 9: This mapping rule replaces the original web service invocation by an ISReal object service invocation.

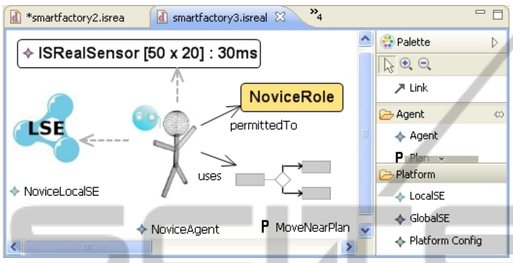


Figure 10: The customized ISReal agent diagram.

artifacts, the ISReal sensor and the LSE. Placing an ISReal agent implies, compared to a plain agent, the generation of an ISReal agent component which integrates into the ISReal platform, the LSE, sensor interfaces, different service execution semantics, etc.

We evaluated the ISReal extension in a virtual production line scenario. The current MAS model encompasses about 20 goals and plans which cover basic navigation and object interaction (e.g. for operating a virtual machine). Code for simple plans such as the one shown in Figure 6 can be completely generated and do not need further manual refinements at code level. Our experience with modeling the demo scenario shows that the extension mechanism fulfills the requirements of real world applications. The extensions focus on single isolated aspects and integrate into the core DSL and the Jadex base transformation. Sometimes we experienced limitations of the current interfaces. For example, it is currently not possible to have mixed target conditions in goals (e.g. a combined SPARQL-Ask query plus a boolean Java expression). This issue could be solved by expression containers which allow to combine different software languages. However, we will further develop the interfaces as we gain experiences with applying BOCHICA to other execution environments. The existing extensions clearly simplify the development of ISReal agents.

6 RELATED WORK

During the recent years, several agent-oriented modeling languages have been proposed. The majority of the modeling languages were created in order to support a certain agent methodology (Henderson-Sellers and Giorgini, 2005)(Sterling and Taveter, 2009). One problem of existing methodology-oriented modeling approaches that we see is that they do not clearly distinguish between (i) the agent platform, (ii) the methodology, and (iii) the modeling language. Two indicators which support our perception are (i) the development of the modeling languages is not decoupled from the methodologies and (ii) none of the languages has an own name (only the tools have names). However, we think the development of modeling languages is orthogonal to the development of agent methodologies and tools. Of course, a methodology can (and most likely will) have certain requirements to a modeling language (e.g. own methodology artifacts and views). For this purpose, BOCHICA can be extended with methodology artifacts. However, the core of the agent modeling language is independent of a certain methodology. Unfortunately, the majority of the developed modeling tools are only partially based on standardized technology for model-driven development[13] which hampers the benefits of MDSD. For example, the *Prometheus Design Tool*[14] (Prometheus methodology) has no explicit underlying metamodel. AgentTool III[15] (O-MaSE), INGENIAS Development Kit[16] (INGENIAS), Taom4e[17] (Tropos), and REBEL[18] (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 Metamodel and Profile*[19] (AMP) or FIPA Agent UML[20]). Those approaches promise to reuse the ecosystem built around UML – including the large user group. However, modeling agents is fundamentally different from modeling objects. Agents possess an internal cognitive 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 com-

[13]We analyzed the publicly available software.
[14]http://www.cs.rmit.edu.au/agents/pdt/
[15]http://agenttool.cis.ksu.edu/
[16]http://ingenias.sourceforge.net/
[17]http://selab.fbk.eu/taom/
[18]http://www.agentlab.unimelb.edu.au/software.html
[19]http://www.omg.org/cgi-bin/doc?ad/08-09-05.pdf
[20]http://www.auml.org/

plex 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 the 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). This would mean to reinvent many things that are already part of BOCHICA. 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 discussed. 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 the base transformations (similar to existing approaches). We see BOCHICA complementary to existing approaches as it provides a clean conceptual framework and interfaces for integration.

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, new methods, and platforms. After we presented our vision on how to apply our framework in Section 3, the extension interfaces were introduced in Section 4. Based on BOCHICA, we showed how to create a model-driven development environment for semantic virtual worlds. We see our approach as a contribution to the unification of the diverse field of agent-oriented modeling and to bridge agent research and concrete software development. In the future we want to integrate existing agent methodologies and work on collaborative modeling of agent-based systems. [21]

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. In *Proc. of the Int. Conf. on Intelligent Agent Technology (IAT'11)*, pages 101–108. IEEE.

[21] Please contact the first author if you need a backup of the referenced documents in the footnotes (14.10.2011).