The BOCHICA Framework for Model-Driven Agent-Oriented Software Engineering

Stefan Warwas[*]

German Research Center for Artificial Intelligence (DFKI),
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
stefan.warwas@dfki.de

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. agents in virtual worlds) as well as to the target execution environment (e.g. a virtual reality platform). This includes the use of specialized concepts of the application domain, software languages (e.g. query languages for reasoning about an agent's knowledge), as well as custom views and diagrams for designing the system. This paper presents a model-driven framework for engineering multiagent systems, called BOCHICA. The framework is based on a platform independent modeling language which covers the core concepts of multiagent systems. In order to better close the gap between design and code, BOCHICA can be extended through several extension interfaces for custom application domains and execution environments. The framework is accompanied by an iterative adaptation process to gradually incorporate conceptual extensions. The approach has been evaluated at modeling agents in semantically-enriched virtual worlds.

1 Introduction

The research field of *Agent-oriented Software Engineering* (AOSE) is concerned with investigating how methods and algorithms 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).

[*] This paper was partially founded by the Saarbrücken Graduate School for Computer Science.

During the recent years, several approaches for modeling agent-based systems have been proposed (e.g. [1]). Although the developed modeling languages are a step into the right direction, they have problems with fulfilling a user's need to efficiently model a custom 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. The better the modeling language covers the target concepts, the less manual customizations are necessary at code level. Moreover, it is desirable to extend the modeling language with new concepts from AI and agent research (e.g. new ways of modeling interaction protocols). This paper proposes a model-driven framework for engineering agent-based systems which addresses the described problems. The framework, called BOCHICA[1], is based on a *Domain Specific Language* (DSL) which covers the main concepts of MAS. It provides several extension interfaces for integrating new concepts, AOSE methods, and 3rd party software languages. The customizations allow the user to tailor the framework to his needs without loosing the integration into a larger framework. The first part of this paper provides background on AOSE (Section 2), introduces the general idea of the BOCHICA framework (Section 3), and provides an overview of the extension interfaces (Section 4). The second part shows how to tailor the framework to the application domain of agents in semantically-enhanced virtual worlds and presents the evaluation results (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 [2]. 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 [2], "*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 [3][4][5]. 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 [3] 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

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

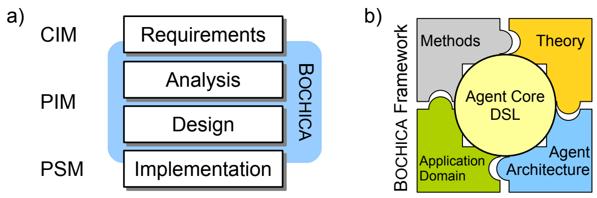


Fig. 1. a) depicts how BOCHICA relates to the abstraction layers defined by MDA (*Computational Independent Models (CIM), Platform Independent Models (PIM), Platform Specific Models (PSM)*). b) visualizes the idea behind the BOCHICA core DSL and 3rd party extensions.

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.

3 Framework Overview

The role of BOCHICA in the overall software development process is to provide the means for capturing the design decisions of a SUC and bridging the gap between design and code. Figure 1 a) depicts how the framework is aligned to the abstraction layers defined by MDA. The agent-oriented modeling language underlying BOCHICA defines the concepts which are available for modeling a SUC (see Figure 1 b). So called *base transformations* are responsible for mapping the concepts of the BOCHICA core DSL to different agent platforms. In real world applications, an agent platform usually does not exist in isolation. As an agent platform is integrated into a larger execution environment, the core DSL gets extended with additional concepts to address the features of that execution environment. Moreover, a so called *extension transformation* defines additional conceptual mappings for the new concepts which complement an existing base transformation. The underlying idea is to reuse large parts of the existing infrastructure. The separation into a core modeling language and 3rd party extensions prevents the core DSL from getting cluttered with highly specialized concepts that are only relevant for a small number of applications (and thus, would make the language unusable over time). In the following, we provide a brief overview of the BOCHICA core DSL and introduce an iterative adaptation process for integrating conceptual extensions into BOCHICA.

3.1 Core DSL

The BOCHICA core DSL is based on the *Domain Specific Modeling Language for Multiagent Systems* (DSML4MAS) [6]. 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

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

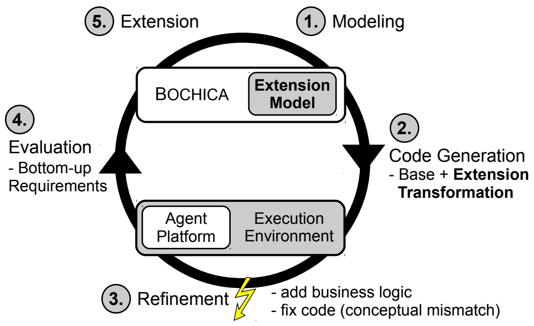


Fig. 2. This figure depicts the process for gradually adapting BOCHICA to a custom application domain

PIM4AGENTS models. Model validation on a platform independent level already prevents many errors in early phases of a project. DSML4MAS inherently possesses three different abstraction layers. The *macroscopic* layer defines the organizational structures of the SUC in terms of abstract goals, roles, interactions, and organizations. The *microscopic* layer defines agent types, behavior templates, concrete goals, etc. The *deployment* layer 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.

3.2 Iterative Adaptation

One of the main reasons in model-driven AOSE which causes design and code to diverge over time is that the modeling language is not expressive enough for capturing all design decisions. This makes extensive manual code refinements necessary at the platform level to address certain features of the target environment. To approach this problem, we propose an iterative adaptation process to gradually tailor BOCHICA to the needs of a custom application domain and execution environment (see Figure 2). In the first iteration, the SUC is modeled using the core DSL of BOCHICA without any conceptual extensions (step 1). An existing base transformation to a considered agent platform is applied (step 2). In step (3), the engineer realizes that some concepts of the execution environment cannot be addressed by the core DSL and/or the existing base transformation. The mismatches are analysed in step (4) and so called *bottom-up requirements* are collected. Finally, the new concepts and extension transformation are introduced in step (5). The second iteration makes use of the made extensions so that the gap between design and code is better closed. The adaptation process continues until the framework fits the needs of the SUC. It is important to note that the idea of the adaptation process is not to introduce arbitrary low-level concepts into BOCHICA. The extensions should complement the core DSL where necessary so that the design decisions can be captured. This task is not trivial and requires background on MDSD as well as on the BOCHICA core DSL.

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 are provided by 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`, and `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 evolve over time and will build a solid foundation for AOSE. One example of how the BOCHICA framework can be extended is the approach for an alternative (declarative) way of modeling interaction protocols presented in [7]. The presented approach extended the `Interaction` concept 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[3] framework and the *Eclipse Modeling Framework* (EMF) [8].

4.2 Data Model

Figure 3 depicts the data model interface of BOCHICA. The core of the data model has been separated from BOCHICA and is based on the Ecore metamodel provided by EMF [8]. 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 data model can be made available within BOCHICA by the concept `EType`. On top of the Ecore metamodel, BOCHICA defines basic data structures such as `Sequence`, `Set`, or `HashMap`. Moreover, the data model interface also provides access to internal types such as `Event` and `Goal`. The internal types are required for accessing BOCHICA model artifacts inside a plan (e.g. the parameters of a goal). The data model can be extended by external plug-ins with specialized data structures. It is also possible to introduce an alternative to the Ecore metamodel for defining data types. Technically, the user defined data structures use the same extension interfaces as in Section 4.1.

[3] `http://eclipse.org/equinox/`

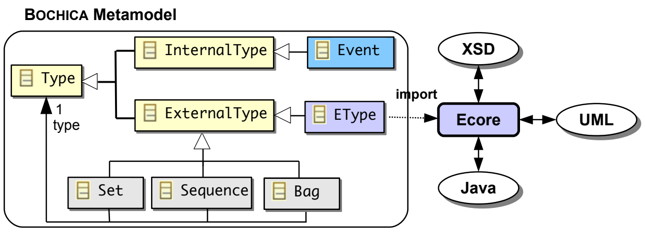


Fig. 3. The BOCHICA data model interface

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 which one to use. BOCHICA 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` and `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[4], Atlantic metamodel zoo[5]). 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 4). 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 an abstract 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 [9]. Most of the developed approaches are supported by a graphical modeling language

[4] `http://www.emftext.org/`

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

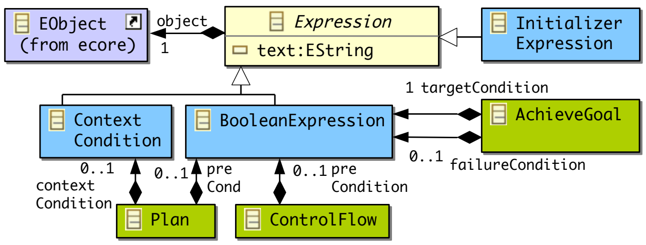


Fig. 4. Expression interface of BOCHICA

(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, BOCHICA can be extended by AOSE method plug-ins to support the software development process. 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 [10] uses so called *scenarios* in the *system specification* phase to identify typical events, actions, and goals of the SUC. In the *architectural design* phase, the collected entities are grouped to roles and agents. A Prometheus plug-in for BOCHICA could extend it with the missing concepts for modeling scenarios 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 `MethodArtifact`. 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 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. 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[6]) 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[7] 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

[6] `http://www.omg.org/spec/QVT/1.0/`
[7] `http://www.eclipse.org/modeling/m2t/`

transformation and the succeeding one supplements the result of the proceeding one. Thus, an external plug-in for BOCHICA usually consists 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[8].

5 Evaluation

This section evaluates the BOCHICA framework in a complex real world case study. As of 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 a simulation platform based on semantic web technology for bringing intelligent behavior into virtual worlds [11]. 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 discuss the BOCHICA extensions for developing intelligent ISReal avatars, 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 pre-condition 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,

[8] `http://www.eclipse.org/gmf`

Jadex, and the Xaitment[9] 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 a XML3D[10]-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[11]-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 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 [11]. The remainder of this section discusses the extensions of the BOCHICA framework for developing ISReal agents.

5.1 Conceptual Extension

Figure 6 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 bottom 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 7 depicts a 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

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

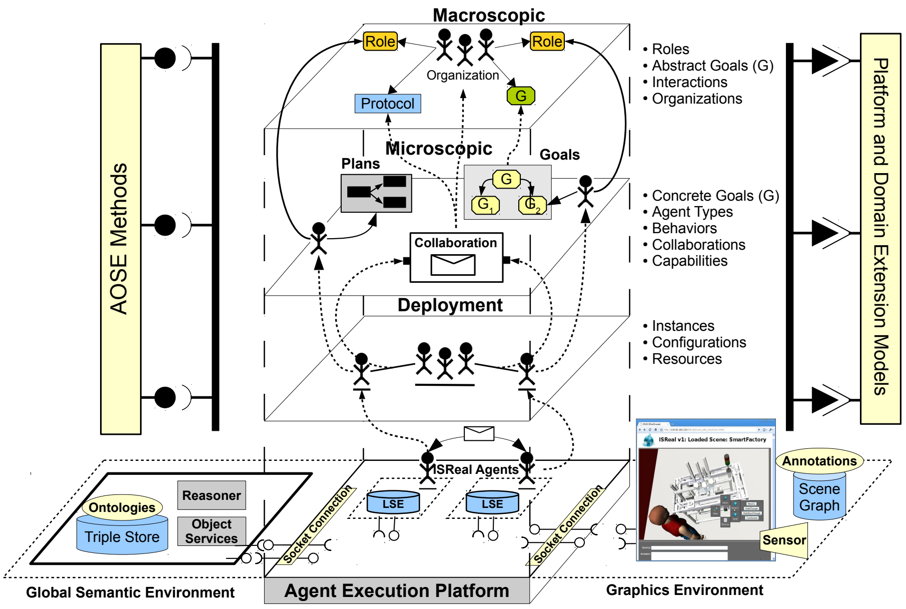


Fig. 5. The bottom layer depicts the components of the ISReal platform. The upper central part shows the inherent degrees of abstraction of BOCHICA. The left an right hand side represent the interfaces for extending the framework.

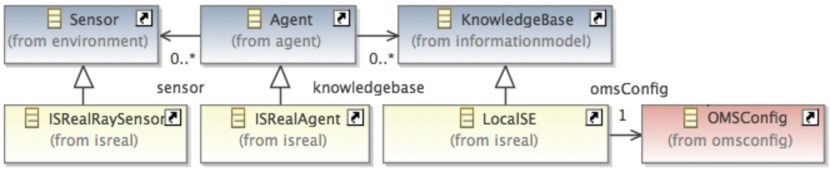
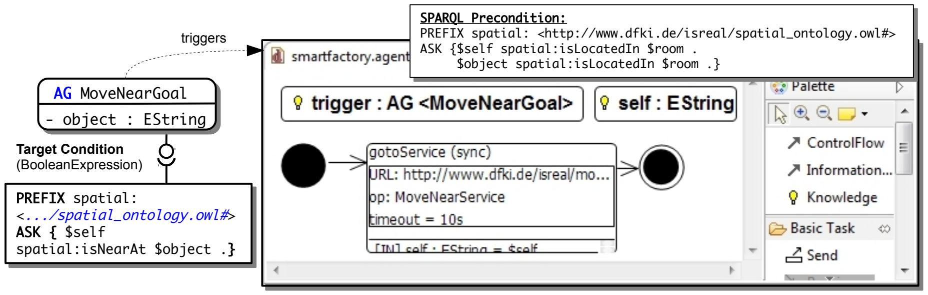


Fig. 6. A part of the ISReal extension of BOCHICA



```
PREFIX spatial:
<.../spatial_ontology.owl#>
ASK { $self
spatial:isNearAt $object .}
```

```
SPARQL Precondition:
PREFIX spatial: <http://www.dfki.de/isreal/spatial_ontology.owl#>
ASK {$self spatial:isLocatedIn $room .
     $object spatial:isLocatedIn $room .}
```

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

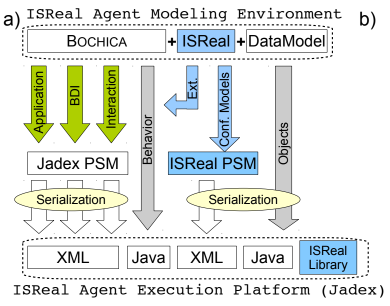
Fig. 7. Agent-based orchestration of ISReal object services (behavior diagram)

rooms). The plan's precondition 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).

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 `isNearAt(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.

a)



b)

```
«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»
```

Fig. 8. a) depicts an overview of the transformation architecture and b) an example mapping rule in XPand

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, BDI, Interaction, and Behavior (see Figure 8 a). 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

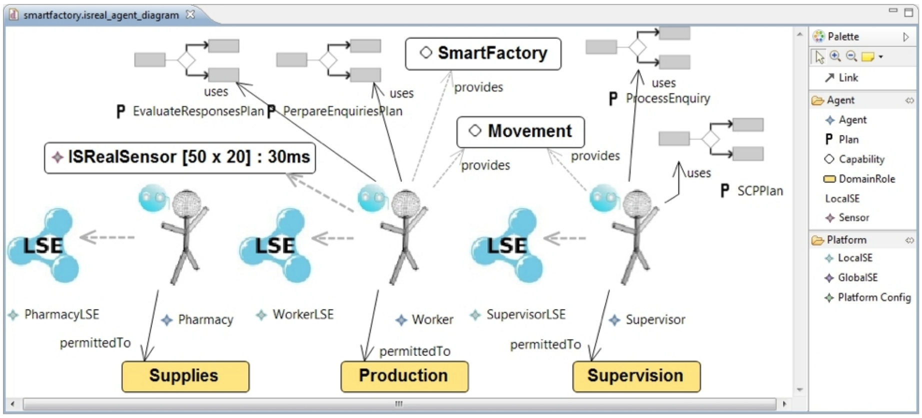


Fig. 9. The customized ISReal agent diagram

Jade (to avoid overhead and simplify extensions). The model-to-text transformation is done using XPand. The separation of the transformation into modules leverages extensibility and eases maintenance. As explained in Section 4.5, the data 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 8 b) 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 9 depicts the ISReal agent diagram which contains, in addition to the standard BOCHICA 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 a)

	BOCHICA	ISReal Extension
Concepts	124	8 + 1 Language
Mapping Rules	207	22
Mapping Rules LOC	4145	726

b)

	Java	XML	Total
Generated	4907	1630	6537
Custom	750	0	750

Fig. 10. Table a) compares the BOCHICA core DSL and the Jadex base transformation with the ISReal-specific extensions. b) compares the generated code for the SmartFactory case study to the manually written *Lines of Code* (LOC).

into the ISReal platform, the LSE, sensor interfaces, different service execution semantics, etc.

The ISReal-enabled BOCHICA framework has been evaluated in a virtual production line scenario. For this purpose, a virtual representation of the DFKI SmartFactory[12] was semantically annotated. The SmartFactory is a real existing machine of the DFKI to evaluate technology for the factory of the future. The BOCHICA ISReal extension has been used to model agents that perform typical workflows such as handling new orders and fixing problems as they occur. Figure 9 depicts an overview of the modeled ISReal agents. The current model encompasses basic navigation and object interaction (e.g. for operating the machine). In order to estimate the effort for the ISReal customization, Figure 10 depicts two tables. Table a) compares (i) the number of concepts, (ii) the number of transformation rules, and (iii) the *Lines of Code* (LOC) of the model transformation between BOCHICA and the ISReal extension. Table b) compares the generated code with the manually written code for the SmartFactory scenario. The manual code mainly implements business logic (e.g. computing the shortest path between two objects). The ISReal extension required around 10% new concepts and mapping rules. The result is a modeling environment which addresses the needs of one specific application domain. BOCHICA is especially suited for large and medium size application domains and 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). Probably the biggest obstacle of the approach is the required additional knowledge of the BOCHICA core DSL and MDSD.

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 [1] [9]. 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

[12] `http://smartfactory.dfki.uni-kl.de/de/`

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 which hampers the benefits of MDSD. For example, the *Prometheus Design Tool*[13] (Prometheus methodology) has no explicit underlying metamodel. Others like AgentTool III[14] (O-MaSE), INGENIAS Development Kit[15] (INGENIAS), Taom4e[16] (Tropos) are only partially based on MDSD (e.g. proprietary or non-MDA-based model transformations). To the best of our knowledge, the mentioned approaches do not consider extensibility as presented by this paper. 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*[17] (AMP) or FIPA Agent UML[18]). 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 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 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 [12] two platform specific modeling languages for the agent platforms SEAGENT [13] 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.

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

[13] `http://www.cs.rmit.edu.au/agents/pdt/`
[14] `http://agenttool.cis.ksu.edu/`
[15] `http://ingenias.sourceforge.net/`
[16] `http://selab.fbk.eu/taom/`
[17] `http://www.omg.org/cgi-bin/doc?ad/08-09-05.pdf`
[18] `http://www.auml.org/`

on a platform independent agent core DSL and provides generic extension interfaces for integrating results from agent research as well as for customizing it regarding user-specific application domains, AOSE methods, and target platforms. Section 3 provided an overview of BOCHICA and presented an iterative adaptation process for integrating conceptual extensions. The framework interfaces were introduced in Section 4. We evaluated BOCHICA at the application domain of agents in semantically-enhanced virtual worlds. Our experience showed that around 10% of new concepts and mapping rules were necessary to create a development environment which enables efficient modeling of ISReal agents. 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.

References

1. Henderson-Sellers, B., et al.: Agent-Oriented Methodologies. IGI Global (2005)
2. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley Longman, Amsterdam (2008)
3. Belecheanu, R.A., et al.: Commercial applications of agents: Lessons, experiences and challenges. In: 5th Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems, pp. 1549–1555 (2006)
4. Jennings, N.R., Wooldridge, M.: Agent-Oriented Software Engineering. Artificial Intelligence 117, 277–296 (2000)
5. McKean, J., et al.: Technology diffusion: analysing the diffusion of agent technologies. Autonomous Agents and Multi-Agent Systems 17, 372–396 (2008)
6. Hahn, C., et al.: A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems 18, 239–266 (2009)
7. León-Soto, E.: A Model-Driven Approach for Executing Modular Interaction Protocols Using BDI-Agents. In: Fischer, K., Müller, J.P., Levy, R. (eds.) ATOP 2009 and ATOP 2010. LNBIP, vol. 98, pp. 10–34. Springer, Heidelberg (2012)
8. Steinberg, D., et al.: EMF: Eclipse Modeling Framework, 2nd revised edn. Addison-Wesley (2008)
9. Sterling, L., et al.: The Art of Agent-Oriented Modeling. The MIT Press (2009)
10. Padgham, L., et al.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons (2004)
11. 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)
12. Kardas, G., Ekinci, E.E., Afsar, B., Dikenelli, O., Topaloglu, N.Y.: Modeling Tools for Platform Specific Design of Multi-Agent Systems. In: Braubach, L., van der Hoek, W., Petta, P., Pokahr, A. (eds.) MATES 2009. LNCS, vol. 5774, pp. 202–207. Springer, Heidelberg (2009)
13. Dikenelli, O.: SEAGENT MAS platform development environment. In: Proc. of the 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems: Demo Papers, AAMAS 2008, pp. 1671–1672. IFAAMAS (2008)