Making Multiagent System Designs Reusable: A Model-driven Approach

Stefan Warwas, Matthias Klusch
German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany
{*stefan.warwas, matthias.klusch*}*@dfki.de*

***Abstract*—Software engineers usually approach complex problems by separating them into sub-problems. In multiagent systems, sub-problems are solved by autonomous agents that build organizational structures. Each agent internally further decomposes a problem by goal and plan hierarchies. The underlying design of a system reflects an engineers experience with approaching complex problems. It is desirable to reuse patterns and structures that proved their practical use and were validated (e.g. interaction protocols, goal hierarchies, behavior templates, organizational structures, etc.). This improves software quality and reduces development time and costs in the long run. Model-driven software engineering enables the separation of the platform independent design of the system under consideration and the actual implementation for a concrete execution platform. The gap between the different levels of abstraction is closed by model transformations. In this paper we propose a model-driven reverse engineering approach for lifting the underlying design of implemented multiagent systems to a platform independent level. For this purpose we specify conceptual mappings from the platform to our platform independent modeling language. The extracted structures have to be refined (e.g. by using existing platform independent artifacts) and can be re-used as blue print for solving similar problems on similar execution platforms. We evaluated our approach for the BDI agent platform Jadex and a real-world scenario.**

***Keywords*-Agent-oriented Software Engineering, Model-driven Reverse Engineering, PIM4Agents, Jadex**

I. Introduction

Today's software systems are becoming increasingly complex. Software engineers have to deal with heterogeneous, dynamic, and distributed IT environments. At the same time reducing development costs and increasing software quality are two of the main challenges. *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). Even if MDSD made much progress, most code has already been written without MDSD in mind and very often models and code diverge over time.

Agent-oriented Software Engineering (AOSE) is a novel software engineering paradigm which promises to tackle the complexity of problems in distributed environments more elegantly than established approaches. However, AOSE is still driven by research and has not reached the same level of maturity than *Object-Oriented Software Engineering* (OOSE). To make AOSE a real alternative to OOSE new methods and tools have to be developed. In this paper we propose a *Model-Driven Reverse Engineering* (MDRE) approach for making the underlying design of *Belief, Desire, Intention* (BDI) [1] agents reusable on a platform independent layer. We use a domain specific modeling language for representing the different artifacts and storing them in a model repository (see Figure 1). For this purpose we specify mappings between the platform concepts and the platform independent language. Finally, we apply manual refinements to the lifted models. We evaluated our approach for the BDI agent platform Jadex[2]. We chose Jadex because it is open source and has an active community.

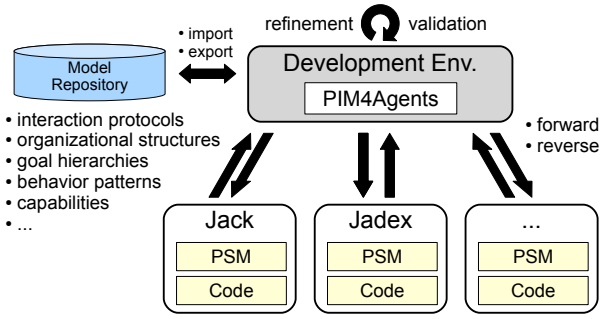

Figure 1. Overview of our approach for reusing multiagent system designs.

This paper is structured as follows: Section II provides background on our language-driven approach in general. Section III introduces the metamodels which build the basis for our MDRE approach. Section IV presents the conceptual mappings from Jadex to PIM4AGENTS. Section V shows

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

[2]http://jadex.sourceforge.net

how to manually refine the lifted models. Afterwards, Section VI evaluates the approach on a real world Jadex system. Section VII provides an overview of related work and finally Section VIII concludes this paper.

II. Domain Specific Languages

Domain Specific Languages (DSLs) - in opposite to *General Purpose Languages* (GPLs) - are designed for a certain domain and purpose. Thus, they usually cover this domain more efficiently than GPLs (e.g. UML or Java). DSLs can be on different levels of abstraction. In opposite to platform specific languages, platform independent languages neglect certain details which are not necessary for the considered level of abstraction. Abstraction is necessary for handling the steadily increasing complexity of today's software systems [2]. In general, a DSL is defined by (i) an abstract syntax, (ii) concrete syntax, and (iii) semantics.

Our approach is based on the *Domain Specific Modeling Language for Multiagent Systems* (DSML4MAS) [3]. 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*[3] (OCL)-based constraints for validating PIM4AGENTS models. Model validation on a platform independent level already prevents many errors in early phases of a project. It is also important for building model repositories with validated artifacts which can be reused. In previous work we specified conceptual mappings between PIM4AGENTS and the concepts of the execution platforms Jack[4] and Jade[5] which are used for forward engineering [3].

As already mentioned, PIM4AGENTS is platform independent but it possesses different *degrees of abstraction* (see Figure 2). The *requirements layer* is the most abstract degree and covers abstract goals, roles, interactions, and organizations. The *system design* degree contains (i) agent types, (ii) behavior templates, (iii) concrete goals, etc. The lowest degree is the *deployment* layer which specifies concrete deployment configurations (e.g. agent instances and resources). Our development environment also offers specialized views for modeling the various aspects and their interdependencies. Although PIM4AGENTS is platform independent, it is not architecture independent. What differs agents from objects, services, etc. is their internal architecture (the underlying mental model). Thus, to exploit the benefit of AOSE, an agent-oriented modeling language has to consider the agent

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

[4]http://aosgrp.com/products/jack/index.html

[5]http://jade.tilab.com/

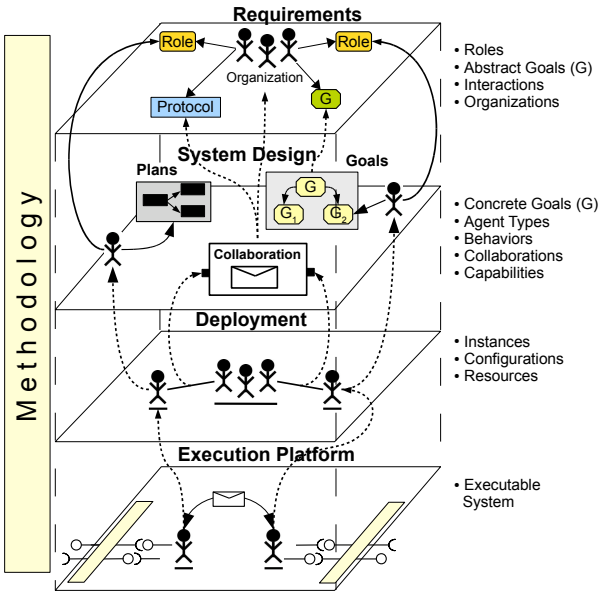

Figure 2. The bottom layer represents the execution platform. The upper layers are the degrees of abstractions in DSML4MAS.

architecture as first class object. To summarize all properties: DSML4MAS is a platform and methodology independent, (BDI) architecture aware, graphical modeling language with several inherent degrees of abstraction and aims on main stream large scale agent-oriented software development. Those properties make DSML4MAS the perfect language for reusing BDI designs.

III. Metamodels

Metamodels specify the concepts and their relations which are available for modeling a SUC. Now, we introduce the metamodels which are necessary for MDRE of Jadex applications. The Jadex metamodel is introduced in Section III-A. Afterwards, Section III-B presents the PIM4AGENTS metamodel. Due to the limited space we can only introduce representative parts.

A. Jadex

A Jadex application is implemented using XML files which specify agents, capabilities, etc. and Java-based behaviors (see Figure 3). To enable Jadex for model-driven development, we created a Jadex metamodel which is based on the official Jadex XML schema files[6]. Furthermore, we used a Java metamodel for lifting Java-based Jadex plans to the model level. Finally, we created a Jadex *Project* metamodel which aggregates all resources of a Jadex application (e.g. agents, capabilities, and plans).

Jadex Application Metamodel. The concept `Applicationtype` is the main container of a Jadex application and

[6]Our Jadex metamodel covers all aspects of Jadex. Due to space restrictions we neglect the agrs- and envspace metamodels.

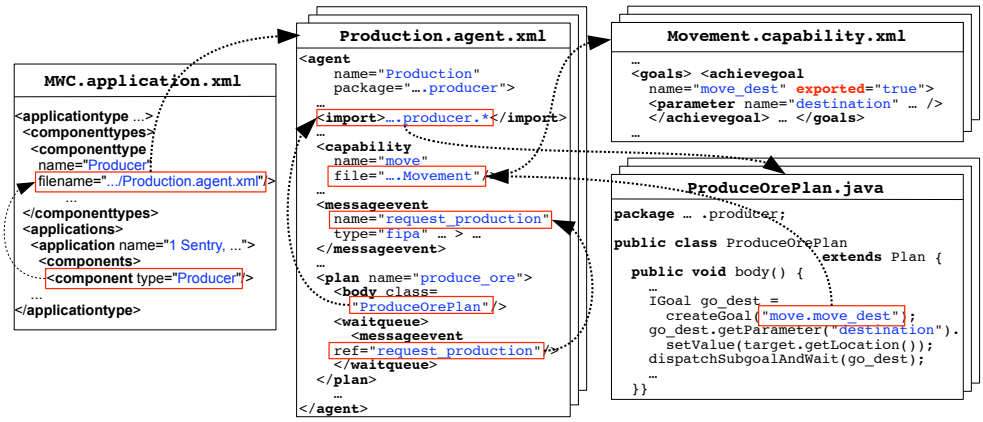

Figure 3. This figure visualizes the basic structure of a Jadex project. The code is taken from the "Mars World Classic" example of the official Jadex distribution. The arrows highlight the interdependencies between XML-based application, agent, and capability files and Java-based plans.

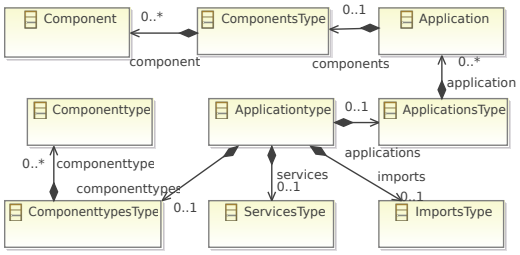

Figure 4. Main concepts of the Jadex application metamodel.

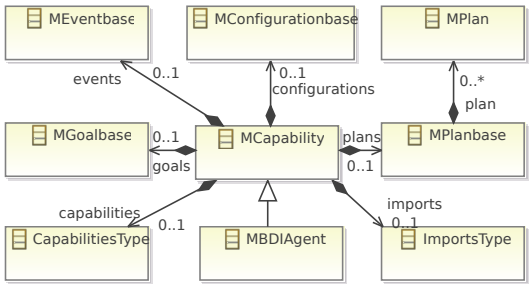

Figure 5. Main concepts of the Jadex BDI metamodel.

possesses a `name` and `package` (see Figure 4). Resources that are not in the current scope have to be imported by the `Import` concept. All components (such as agents) have to be declared by the concept `Componenttype` which has a `filename` attribute to reference Jadex agent XML files. The concept `Application` defines a single Jadex application and contains a set of `Component` declarations. A `Component` is an instance of the declared `Compenttype` and can refer to a certain configuration (initialization of beliefs, goals, etc.) that should be used. Configurations are declared in a component's XML file.

Jadex BDI Metamodel. The Jadex BDI metamodel specifies the concepts which are necessary to define agents and capabilities (see Figure 5). The main concepts are `MBDIAgent` and `MCapability`. The only difference between Jadex agents and capabilities is that agents possess an own execution thread. A `MCapability` imports several resources by the concept `ImportsType`. Moreover, the `MCapability` declares several events (e.g. `MInternalEvent` and `MMessageEvent`) which are contained by `MEventbase`. Likewise, goal declarations are contained by the concept `MGoalbase`. The top-most concept for all goals is `MGoal`. There are four concrete goal types in Jadex: `MAchieveGoal`, `MPerformGoal`, `MMaintainGoal`, and `MQueryGoal`. Used plans are declared by the concept `MPlan` which is contained by `MPlanbase`. A `MPlan` contains a `MPlanBody` which refers to a Java-based Jadex plan. The `MPlanTrigger` concepts refers to the triggering event of the plan. Finally, a `MCapability` can contain several `MConfigurations` which are contained by `MConfigurationbase`. A configuration initializes an agent with certain beliefs and goals.

Java Metamodel. Since we want to map Jadex plans and information models (which are implemented in Java) to PIM4AGENTS, we need a metamodel for Java. Because the creation of a Java metamodel and the lifting from code to model is a big endeavour on its own, we rely on the Java metamodel and tool support provided by the Eclipse *MoDisco* Project[7]. MoDisco is part of the *Eclipse Modeling Framework*[8] (EMF) which is a framework for MDSD based on MDA standards and Eclipse technology.

The top-most container for a set of Java files is the concept `Model` (see Figure 6). The type of a `ClassFile` is specified by an `AbstractTypeDeclaration`. An `AbstractTypeDeclaration` contains a set of `Body-`

[7]http://wiki.eclipse.org/MoDisco/Java
[8]http://www.eclipse.org/emf

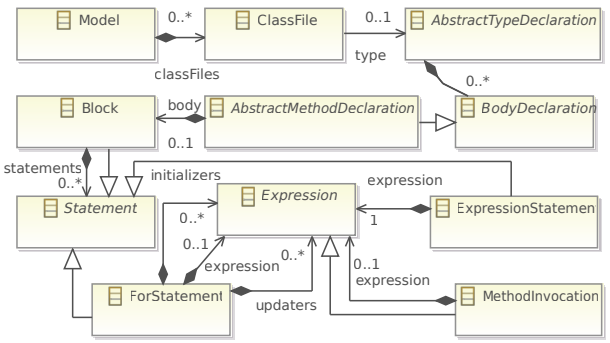

Figure 6. Main concepts of the MoDisco Java metamodel.

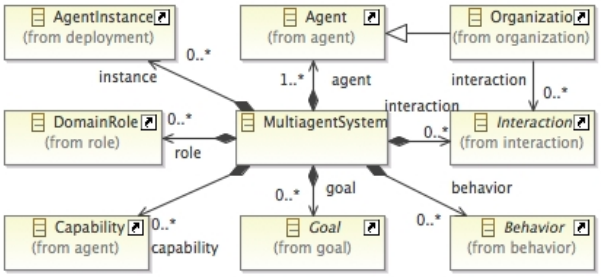

Figure 7. The central concepts of the PIM4Agents metamodel.

`Declarations` which consists of a (code-) `Block`. `Statements` are contained by `Block`. There are many different kinds of `Statements`. For example, a `ForStatement` has a (condition-) `expression`, an `initializer` expression, and an `updater` expression. A `MethodInvocation` is also an `Expression`. In order to use the generic Java metamodel for reverse engineering Jadex behaviors the Jadex API has to be taken into consideration. For example, sending a message in Jadex is done by invoking the `sendMessage` method of the API. The implementation of a mapping rule has to check whether a generic Java `MethodInvocation` is a call to the Jadex API for sending a message. Thus, the Java metamodel plus the knowledge of the Jadex API is the conceptual basis for the mapping rules.

Project Metamodel. Since each XML file (agent, capability, etc.) will be lifted to a Jadex model, we have to deal with a set of models. Thus, we introduce a new Jadex project metamodel. The `Project` concept is a container which aggregates the different root elements of the imported metamodels (e.g. `Applicationtype`, `MBDIAgent`, `Model`).

B. PIM4Agents

The PIM4AGENTS metamodel is the target metamodel for our reverse engineering approach. We focus on the multiagentsystem, behavior, and goal aspects of PIM4AGENTS because they are central for the understanding of the conceptual mappings presented later. For a comprehensive introduction to PIM4AGENTS we refer to [3]. Figure 7 depicts the MAS aspect of PIM4AGENTS. The concept `MultiagentSystem` is the top-most container and consists of agent and organization specifications, interaction protocols, behaviors, goals, capabilities, etc.

PIM4Agents Behavior Metamodel. A PIM4AGENTS `Plan` consists of a set of `Activities` that are connected by `ControlFlows`. There are two types of activities: (i) `StructuredAcitivities` (e.g. loop or decision) consists of a set of sub-`Activities` and (ii) `Tasks` are atomic activities that cannot be further decomposed. For example, sending and receiving messages (concepts `SendTask` and `ReceiveTask`), posting goals (concept `AssumeGoal`) are `Tasks`. `InternalTasks` are used to encapsulate algorithms or library calls that are not considered at the platform independent layer. `ControlFlows` determine the execution order of `Activities`. Information is stored by the `Knowledge` concept. `Knowledge` has a name and type.

PIM4Agents Goal Metamodel. A PIM4AGENTS `Goal` can be either of type `AbstractGoal` or `ConcreteGoal`. `AbstractGoals` are used to support a gradual refinement from abstract requirements to the concrete MAS design. An `AbstractGoal` is realized by a `ConcreteGoal` which inherits from `Event`. There are five types for concrete goals: `AchieveGoal`, `PerformGoal`, `QueryGoal`, `MaintainGoal`, and `MetaGoal`. The goal hierarchy is explicitly represented through `DecompositionLinks` which can be `AndDecompositionLink` or `OrDecomposition-Link`. A `Goal` can have several parameters of type `Knowledge`. `Goals` can also have explicit conflicts among each other.

IV. CONCEPTUAL MAPPINGS

After we introduced the required metamodels, we now specify the mapping rules (MR) from Jadex and Java concepts to PIM4AGENTS. Basically, the Jadex project and application metamodel covers similar concepts to the PIM4AGENTS MAS aspect, the PIM4AGENTS behavior aspect corresponds to the Jadex Java metamodel, and the PIM4AGENTS goal aspect covers similar concepts to the Jadex BDI metamodel. We structured the mappings into application mappings, agent and capability mappings, behavior mappings, event and goal mappings, and information model mappings. In the remainder of this Section we use following abbreviations for the metamodels: PROJ (Jadex Project), APP (Jadex Application), BDI (Jadex BDI), JAVA (Java), and P4A (PIM4AGENTS).

A. Application Mappings

MR-0: $PROJ : Project \rightarrow P4A : MultiagentSystem$
The entry point of the reverse transformation is the mapping from `Project` to `MultiagentSystem` and gets

$APP : Applicationtype \rightarrow P4A : MultiagentSystem$		
target	source	MR
name	The name of the Jadex `Applicationtype` is mapped to the name of the PIM4AGENTS `MultiagentSystem`.	–
agent	Each `Componenttype` declaration of a Jadex `Applicationtype` is resolved to the actual `MBDIAgent`. The resolved `MBDIAgent` is mapped to a PIM4AGENTS `Agent`.	3
behavior	Each Jadex Java plan class (concept `AbstractTypeDeclaration`) which is used by an agent (concept `AgentType`) or capability (concept `MCapability`) of the Jadex application is mapped to a PIM4AGENTS `Behavior`.	5
interaction	Jadex has no explicit representation of interaction protocols. Thus, we map all `MMessageEvents` declared by Jadex agents or capabilities to PIM4AGENTS `Messages` (for later manual refinement; see Section V).	10
capability	Each capability (concept `MCapability`) which is used in the Jadex application is mapped to a PIM4AGENTS `Capability`.	4
goal	Each goal declaration (concept `MGoal`) of all agents and capabilities of the Jadex application is mapped to a PIM4AGENTS `Goal`.	9
instance	Each `Component` which is contained by the user specified `Application` is mapped to a PIM4AGENTS `AgentInstance`.	2

Table I
DETAILS OF THE APPLICATION MAPPING (MR-1).

the two user specified parameters `$applicationName` and `$configurationName`. Both parameters are used to select a certain Jadex application and configuration for the transformation (there might be several). MR-1 is applied to map the user-specified `Applicationtype` to `MultiagentSystem`.

MR-1: $APP : Applicationtype \rightarrow P4A : MultiagentSystem$
This rule maps a concrete Jadex application to a PIM4AGENTS model. The `Applicationtype` concept is the top-most container of a Jadex application and is mapped to a PIM4AGENTS `MultiagentSystem` which is the root element of a PIM4AGENTS model. Table I depicts the details of this mapping rule.

MR-2: $APP : Component \rightarrow P4A : AgentInstance$
Concrete agent instances in a Jadex application are specified by the `Component` concept. A component's `type` attribute refers to a `Componenttype`. The `filename` attribute of the `Componenttype` is used to resolve the concrete Jadex agent model (concept `MBDIAgent`). The resolved type is used as the type of the `AgentInstance` in PIM4AGENTS.

B. Agent Mappings

MR-3: $BDI : MBDIAgent \rightarrow P4A : Agent$
This mapping rule maps a Jadex `MBDIAgent` to a PIM4AGENTS `Agent`. The basic structure of both concepts is similar. An agent's `MBeliefs` are mapped by MR-11, each `MCapability` by MR-4, and each `MPlan` is resolved to the Java class and mapped by MR-5. Additionally, we create a default `DomainRole` which is performed by the agent. The `DomainRole` has the same name as the agent.

MR-4: $BDI : MCapability \rightarrow P4A : Capability$
The declared beliefs and plans of a Jadex `MCapability` are mapped to PIM4AGENTS knowledge and plans, respectively (see MR-11 and MR-5). The name of the source object is assigned to the name of the target object.

C. Behavior Mappings

There exist various concepts in the Java metamodel which have to be mapped to PIM4AGENTS. Due to space restrictions we focus on the concepts `Class` (plan), `MethodInvocation`, `Statement`, and `ForStatement`.

$Java : AbstractTypeDeclaration \rightarrow P4A : Plan$		
target	source	MR
name	The name of the Java `AbstractTypeDeclaration` is mapped to the name of the PIM4AGENTS `Plan`.	–
localKnowledge	Each `VariableTypeDeclarationSatement` is mapped to a PIM4AGENTS `Knowledge`.	11
steps	The plan class's `body()` method is resolved (concept `AbstractMethodDeclaration`) and all contained Java `Statements` are mapped to corresponding PIM4AGENTS `Activities`.	6, 7, 8
controlFlow	The Java `Statement` of the plan's `body` method have a certain order. The generated PIM4AGENTS `Tasks` and `StructuredActivities` are connected with `ControlFlows` in the same order as the `Statements` they were created from.	–

Table II
DETAILS OF THE PLAN MAPPING (MR-5).

MR-5: $JAVA : AbstractTypeDeclaration \rightarrow P4A : Plan$
This rule maps a Java `AbstractTypeDeclaration` which inherits from the Jadex `Plan` class to a PIM4AGENTS `Plan`. Table II depicts the details of this mapping rule.

MR-6: $JAVA : MethodInvocation \rightarrow P4A : AssumeGoal$
A helper function is used to check whether a Java `MethodInvocation` is a call to the Jadex `dispatchSubgoal()` or `dispatchSubgoalAndWait()` methods. If this is the case, the method invocation is mapped to either a PIM4AGENTS `AssumeGoal` or `AssumeGoalAndWait` task. The posted goal type is resolved and assigned to the `AssumeGoal` task.

MR-7: $Sequence(JAVA : Statement) \rightarrow P4A : InternalTask$
A sequence of Java `Statements` which cannot be mapped to a certain PIM4AGENTS `Task` or `StructuredActivity` is mapped to an `InternalTask`. An `Internal-`

`Task` is a black box which encapsulates business logic. All variables which are accessed by the enclosed code are added as input/output parameters to the `InternalTask` (see MR-11).

MR-8: $JAVA : ForStatement \rightarrow P4A : Loop$
A `ForStatement` is directly mapped to a `Loop` in PIM4AGENTS. The declared variables are mapped to `Knowledge` (see MR-11). `Begin` and `End` tasks are created for the loop (see Section VI for an example). The contained Java `Statements` are mapped by MR-6 to MR-8 and connected with `ControlFlows`.

D. Event and Goal Mappings

MR-9: $BDI : MGoal \rightarrow P4A : Goal$
The goal's `name` is mapped to the PIM4AGENTS goal name. The `MParameter` elements which are contained by the goal are mapped to PIM4AGENTS `Knowledge` which are parameters of the PIM4AGENTS goal (see MR-11). Goal hierarchies are not explicitly represented in Jadex. We use a design extraction algorithm which computes the Jadex goal hierarchy (and/or decomposition). The computed subgoals of an `MGoal` are assigned to a PIM4AGENTS goal's `subgoalLinks` property. Conflicts between goals are represented by the relation `conflicts`. The concrete Jadex goal types nicely match the goal types in PIM4AGENTS (e.g. perform goal or achieve goal).

MR-10: $BDI : MMessageEvent \rightarrow P4A : Message$
In opposite to Jadex, PIM4AGENTS has an explicit representation of interaction protocols. Since the information is missing in Jadex, the mapping is difficult. We map a Jadex `MMessageEvent` to a PIM4AGENTS `Message`. The name of the message is assigned to the PIM4AGENTS `Message`. We propose to use an already existing interaction protocol from the model repository to manually refine the lifted `Messages` (see Section V).

E. Information Model Mappings

In DSML4MAS, the information model has been separated from the actual PIM4AGENTS metamodel. As information model we use the Ecore metamodel of the EMF framework [4]. This 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. Thus, we see the lifting of the information model from Java classes to the Ecore metamodel as a separate task which is supported by EMF (see [4] for details). There are several possibilities how to use the information model in our reverse engineering approach: (i) the information model can be manually created for small projects using the Ecore modeling tool, (ii) it can be imported from UML, XSD, or Java files using EMF. If no information model is created, no variable types will be assigned in PIM4AGENTS and have to be refined later.

MR 11: $BDI : Belief, BDI : Parameter, JAVA : VariableDeclarationStatement \rightarrow P4A : Knowledge$
A `Belief`, `Parameter`, or `VariableTypeStatement` is mapped to a PIM4AGENTS `Knowledge`. The name and type of the elements are mapped one-to-one to the knowledge's name and type.

V. MANUAL REFINEMENT

Jadex and DSML4MAS share a conceptual core because both cover BDI agents. However, Jadex allows more fine grained control (e.g. about the goal life cycle or by using Java-based plans). As described in Section II, DSML4MAS unifies high level concepts such as roles and abstract goals and concrete agent architecture and deployment aspects. Several high-level concepts like domain roles, interaction protocols, or organizational structures cannot be lifted since they are not explicitly represented in Jadex. Missing information can be (i) manually added from scratch, (ii) reconstructed using a design extraction algorithm (e.g. computing the goal hierarchy), (iii) refined using validated and tested artifacts from the model repository. In this section we want to outline how artifacts from the model repository can be used to refine a lifted model.

Interactions in PIM4AGENTS are at the highest level of abstraction (see Figure 2) and only specify message sequences between interaction roles (see Figure 8). They do not specify the actual role fillers, content types, or time out values. Those details are specified at the system design layer where the protocol is instantiated regarding a certain scenario (concept `Collaboration`). The concrete messages have a link to the abstract interaction messages. What we extract in MR-10 from Jadex are exactly those concrete messages. However, the messages are not explicitly linked to a protocol. Thus, the refinement consists of linking the lifted concrete messages to abstract messages of a corresponding interaction protocol that has been imported from the model repository. The additional information adds expressiveness to the model and can be used to better validate agent systems in large scale applications.

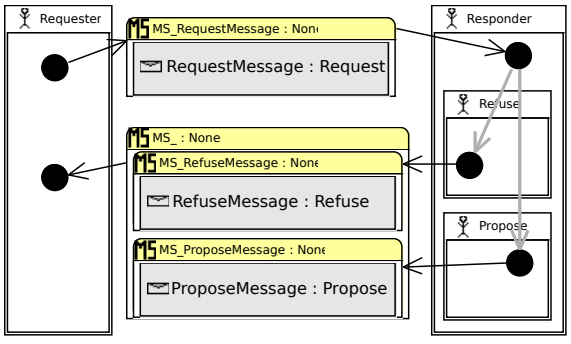

Figure 8. The interaction diagram of the request response protocol.

VI. Evaluation

We evaluated our reverse engineering approach with the *mars world classic* (MWC) example application from the official Jadex 2.0 RC6 distribution. The goal of the MWC agents is to produce ore. There are three agent types. The *sentry* agent's task is to find new resources that can be exploited and creates new tasks for the *production* agent. The production agent produces ore and calls the *carry* agent who brings the ore to the home base. The first task is to create the Jadex project model which contains all resources of a Jadex example project (see Section III-A). A fully automatic *model extractor* iterates over all resources of the Jadex project and uses EMF's and MoDiscos' capabilities to lift Jadex XML and Java files to the model level. All found resources are aggregated in a project model (see Section III). The project model and the information model(s) are passed to the Jadex2PIM4AGENTS transformation. The transformation is based on the mapping rules from Section IV and has been implemented with the standardized model-to-model transformation language QVT[9].

Figure 9 depicts a part of the extracted goal hierarchy of the MWC example. The `walk_around` goal conflicts with the `produce_ore` and `carry_ore` goals. The use of the information model can be seen at the variable types of the goal parameters. Figure 10 depicts the extracted *Producer* agent which has three plans, one capability for movement, and performs the default domain role `Producer`. Figure 11 shows how the `ProduceOrePlan` is mapped to a PIM4AGENTS plan. All variables of the Java plan are mapped to `Knowledge` (MR-11). The `go_dest` knowledge points to the `move_dest` goal type and `target` is of type `Target`. A small part of the information model is depicted at the upper right corner of Figure 11. Code blocks are mapped to `InternalTasks`. The `dispatchSubgoalAndWait()` method call is mapped to an `AssumeGoal` task. After the automatic transformation we manually refined the models. First, we manually added an organization to model the collaboration of the agent types (see Figure 12). The organization has the abstract goal to mine ore. We imported the request response protocol (see Figure 8) from the model repository and used it to explicitly specify the communication between the agents inside the organization (e.g. to assign tasks among the agents). Benefits of the refinement are (i) explicit representation of organizations and protocols make the model much more intuitive, (ii) the artifacts can be used for better validating the model, and (iii) we reuse already tested artifacts.

VII. Related Work

In [5], a general overview of MDRE is provided. [6] provides a taxonomy of reverse engineering. Reverse engineering of agents differs from reverse engineering of object-oriented systems because additional agent-oriented artifacts have to be considered (e.g. organizational structures, goals, interaction protocols, mental models). According to [7], the *INGENIAS Development Kit* (IDK) contains a component called *code uploader* which is used for synchronizing INGENIAS models and code that has been generated from those models (forward engineering). No further information is provided how this component works or which aspects are covered. Similar to IDK, the *Prometheus Development Tool* (PDT) offers code synchronization functionality for generated code that is not further detailed [8]. Both, IDK and PDT support only forward engineering. To the best of our knowledge, there exists no MDRE approach for MAS. In general, there exists not much work regarding agents and reverse engineering. A. Hirst proposes a reverse engineering approach for Soar[10] agents [9]. He focuses on reverse engineering of production rules. In [10], an approach for software comprehension was presented. The idea was to observe agents at run-time (their execution traces) and reconstruct their structures. Agent-oriented methodologies also do not cover reverse engineering. An extensive overview of agent-oriented methodologies is provided by [11].

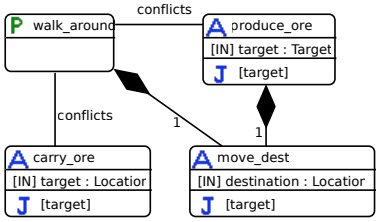

Figure 9. The PIM4AGENTS goal diagram depicts a part of the extracted goal hierarchy (perform and achieve goals) of the MWC example. The black diamonds are and-decomposition links.

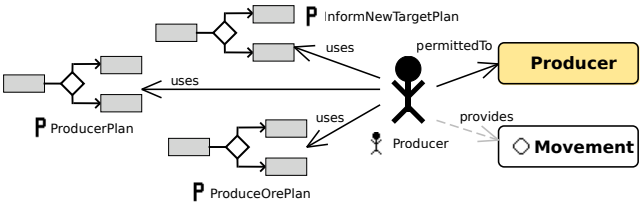

Figure 10. PIM4AGENTS agent diagram of the MWC example (partially).

VIII. Conclusion

In this paper we presented a novel model-driven reverse engineering approach for making the underlying designs of concrete implemented multiagent systems reusable on a platform independent level. First, we introduced the involved metamodels. Afterwards, we presented conceptual mappings from Jadex and Java to PIM4AGENTS. We showed how missing information can be (i) manually refined, (ii) computed with design extraction algorithms, and (iii) refined

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

[10]http://sitemaker.umich.edu/soar/home

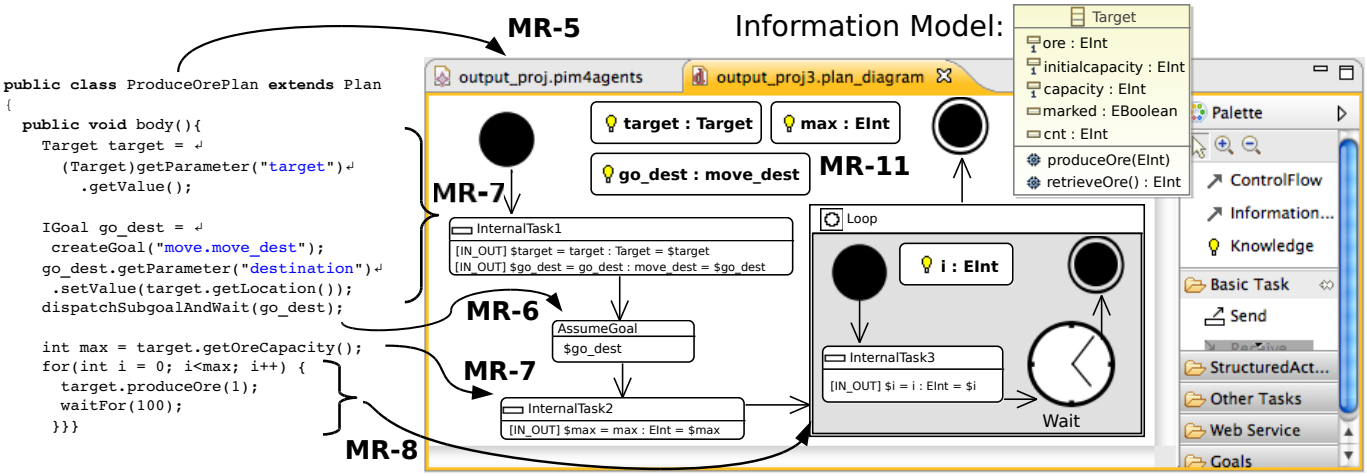

Figure 11. Mapping of a Jadex plan to a PIM4AGENTS behavior (used MRs are depicted). The upper-right corner shows a part of the information model.

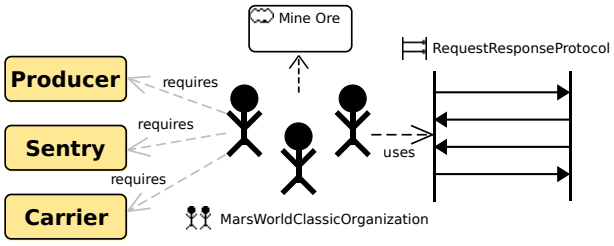

Figure 12. This figure depicts the manually refined MWC organization. It has three domain roles, uses the request response protocol for communication, and has the abstract goal to mine ore.

using artifacts from the model repository. Finally, the approach has been evaluated on a real world application and was integrated into our model-driven development environment [12]. Even if we demonstrated our approach on the Jadex platform, it is generic enough to apply it to similar technology stacks. For this purpose, two things have to be considered: (i) the conceptual gap between the platform and PIM4AGENTS and (ii) the used technology. Mapping rules for other BDI systems will look similar to those of Jadex since they share a conceptual core. On the technical side we showed how a general purpose metamodel for Java can be used for lifting agent artifacts. So, the same mechanism should work for most agent platforms. Existing approaches focus on the top-down direction. Is is worthwhile to take a closer look at concrete agent platforms. This bottom up direction can provide new insights on what agent modeling languages should look like.

REFERENCES

[1] A. Rao and M. Georgeff, "BDI-agents: from theory to practice," in *Proc. of the 1st Int. Conf. on Multiagent Systems (ICMAS'95)*, 1995, pp. 312–319.

[2] A. Kleppe, *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*, 1st ed. Addison-Wesley Longman, Amsterdam, Dec. 2008.

[3] C. Hahn *et al.*, "A platform-independent metamodel for multiagent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 18, pp. 239–266, April 2009.

[4] D. Steinberg *et al.*, *EMF: Eclipse Modeling Framework*, 2nd ed. Addison-Wesley, Dec. 2008.

[5] L. Favre, *Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution*. Engineering Science Reference, Feb. 2010.

[6] E. J. Chikofsky and J. H. C. II, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, vol. 7, pp. 13–17, 1990.

[7] J. J. Gomez-Sanz *et al.*, "INGENIAS development kit: a visual multi-agent system development environment," in *Proc. of the 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'08)*. IFAAMAS, 2008, pp. 1675–1676.

[8] L. Padgham *et al.*, "AUML protocols and code generation in the Prometheus design tool," in *Proc. of the 6th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'07)*. ACM, 2007, pp. 270:1–270:2.

[9] A. J. Hirst, "Reverse engineering of Soar agents," in *Proc. of the 4th Int. Conf. on Autonomous Agents (Agents'00)*. ACM, 2000, pp. 72–73.

[10] D. N. Lam *et al.*, "Comprehending agent software," in *Proc. of the 4th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'05)*. ACM, 2005, pp. 586–593.

[11] L. Sterling and K. Taveter, *The Art of Agent-Oriented Modeling*. The MIT Press, 2009.

[12] S. Warwas and C. Hahn, "The DSML4MAS development environment," in *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'09)*. IFAAMAS, 2009, pp. 1379–1380.