A Visual Development Environment for Jade

(Extended Abstract)

Stefan Warwas, Christian Hahn, Klaus Fischer
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
{stefan.warwas, christian.hahn, klaus.fischer}@dfki.de

ABSTRACT

Agent-oriented software engineering (AOSE) aims on reducing the complexity of multiagent system (MAS) development. Jade is a famous framework for implementing MAS in Java. This paper proposes the model-driven development environment of the *Domain Specific Modeling Language for Multiagent Systems* (DSML4MAS) as a visual development environment for Jade. We focus especially on how the synchronization between design and code has been solved.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; D.2.6 [**Programming Environments**]: Graphical environments

General Terms

Design

Keywords

DSML4MAS, Jade, development environment, modeling tool, PIM4Agents, model-driven

1. INTRODUCTION

Like software engineering in general, building agent-oriented software normally begins with abstract design of its components using agent-based methodologies like Tropos [2] or modeling languages like Agent UML [1]. Finally, the designed system has to be implemented by utilizing existing agent-oriented programming languages which is often a very complex task.

The *Java Agent DEvelopment Framework*[1] (Jade) is one of the most popular open source implementations of a multiagent execution platform. Jade is written in Java and provides an *Application Programming Interface* (API) for developing multiagent systems (MAS) with Java. However, in its current version, Jade does not provide a visual development environment that abstracts from the source code.

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

Cite as: A Visual Development Environment for Jade, (Extended Abstract), Stefan Warwas, Christian Hahn, Klaus Fischer, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1349–1350

The DSML4MAS *Development Environment* (DDE) is a model-driven development environment based on the *Domain Specific Modeling Language for Multiagent Systems* (DSML4MAS). In this paper we propose DDE as a visual development environment for Jade. We especially want to focus on how we handle the synchronization between models and code which is often neglected by tool supported agent-oriented methodologies. Without such a feature the use of a model-driven approach is limited.

Section 2 provides an overview of the DSML4MAS language. DDE is presented in Section 3. How we solved the code generation issue for DDE to use it as a design tool for Jade is outlined in Section 4. Finally, Section 5 concludes this paper.

2. DOMAIN SPECIFIC MODELING LANGUAGE FOR MULTIAGENT SYSTEMS

Like any domain-specific modeling language, DSML4MAS consists of three important features: The abstract syntax is specified by the PIM4AGENTS metamodel which defines the vocabulary of the language [3]. The concrete syntax defines a graphical notation that can be used to design in a graphical manner. Finally, the semantics of the language has been specified with Object-Z [4] and manually translated to the *Object Constrain Language*[2] (OCL) to make use of the constraints within DDE.

DSML4MAS covers language constructs to specify agents, roles, organizations of agents, interaction protocols, behavior of agents, environmental aspects, and deployment of MAS. The deployment aspect can be used to model concrete instances of agent and organization types that can be deployed, for example, in Jade. Modeling a concrete deployment scenario is optional, meaning that concrete instances can also be introduced at run-time.

3. DSML4MAS DEVELOPMENT ENVIRONMENT

DDE is a model-driven development environment based on DSML4MAS. For each of the different aspects covered by DSML4MAS, DDE provides according diagrams for modeling MAS. As DDE is seamlessly integrated into the Eclipse workbench it directly benefits from developments around Eclipse. For example, DDE uses the EMF[3] validation framework to evaluate OCL constraints to ensure the well-formed-

[2]http://www.omg.org/docs/ptc/03-10-14.pdf
[3]http://www.eclipse.org/modeling/emf/

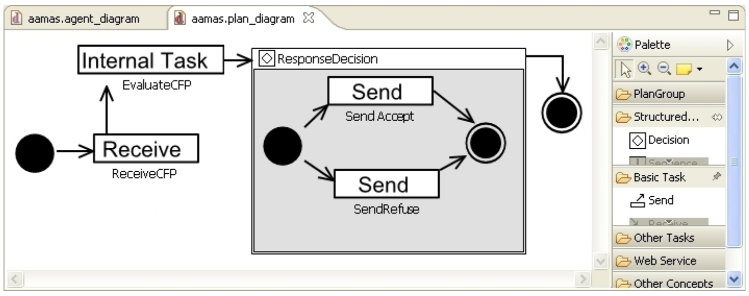

Figure 1: The plan diagram of DDE.

ness of Pim4Agents models. DDE is available under LGPL license and can be found at http://dsml4mas.sourceforge.net.

4. SYNCHRONIZATION OF MODELS AND CODE

After a MAS has been modeled with DDE, the created Pim4Agents model is transformed to Jade Java code. The code generation phase consists of (i) the transformation of the Pim4Agents model to a platform specific model (PSM) for Jade and (ii) the code generation from the Jade PSM to Jade Java code.

The ability to modify the generated source code without worrying that manual changes are overwritten by the regeneration of the source code after changes have been applied to the model is one of the main problems regarding the practical application of model-driven software development. Most tool supported agent methodologies do not address this important issue at all. The solution we outline in this section bases on protected regions in the source code where the user can insert manually written code. We used MOFSCRIPT[4] to implement the code generation.

Figure 1 shows the plan diagram of DDE which can be used to specify the behavior of agents. The depicted plan receives a call for proposal (CFP) message, processes the content, and responds with either an accept or refuse message. Send and receive tasks in Pim4Agents refer to the message that is being sent. An internal task is a kind of black box behavior that is not further refined at the model level. The developer has to implement such a task at the source code level (e.g. an algorithm for evaluating the CFP message).

For every internal task in Pim4Agents the code generation step produces a `OneShotBehaviour` in Jade. Behaviors in Jade have an `action` method which is executed when the behavior is activated. In the body of the `action` method we generate a protected region where the user can insert custom code without worrying about code regeneration. For example, the agent's internal data can be accessed, some existing libraries might be called, or a dialog box can be opened to interact with the user. So, we make sure that design is separated from implementation.

Protected regions can also be used to implement the condition statement of the ResponseDecision from Figure 1. For each *Decision* in Pim4Agents we generate a FSMBehaviour in Jade. For the condition statement itself, we generate a decision-making behavior where the developer can insert custom code in the behavior's `action` method to trigger the different transitions of the FSMBehaviour. Using code of the execution platform to express the condition at the modeling level is not an option since this would break the platform independent character of Pim4Agents models.

The code generator also produces a start class that is used to execute the modeled system in Jade. The deployment aspect of Pim4Agents is used to introduce the modeled agent instances. Protected regions can be used to further customize the setup.

[4]http://www.eclipse.org/gmt/mofscript/

5. CONCLUSIONS

In this paper we proposed DDE as a visual development environment for Jade and outlined a pragmatic way to synchronize design and implementation. The benefits of our approach are that (i) we design reusable components at a platform independent level and (ii) at the same time we are able to produce executable code that (iii) can be modified by the user without worrying about code regeneration. The designed components can be reused (i) for several scenarios and (ii) for several execution platform.

One weakness of our approach is that sometimes there might be better solutions than introducing protected regions in the generated code. However, the outlined approach is practical and significantly increases the practical usability of DDE as a development environment for Jade. Most other tool supported agent methodologies do not address this critical aspect at all. Therefore, we motivate further research into this direction.

6. REFERENCES

[1] B. Bauer, J. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*, volume 1957 of *Lecture Notes in Computer Science*, pages 91–103, Berlin et al., 2001. Springer-Verlag.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[3] C. Hahn. A domain specific modeling language for multiagent systems. In *Proceedings of 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, pages 233–240, 2008.

[4] G. Smith. *The Object-Z Specification Language*, volume 1 of *Advances in Formal Methods*. Kluwer Academic Publishers, 2000.