Unifying JIAC Agent Development with AWE

Marco Lützenberger, Tobias Küster, Axel Heßler, and Benjamin Hirsch

DAI-Labor, Technische Universität Berlin
{marco.luetzenberger|tobias.kuester|axel.hessler|benjamin.hirsch}@dai-labor.de

Abstract. In this paper we describe the Agent World Editor, a tool for designing multi-agent systems and generating executable agent code. The tool also unifies the handling of different agent frameworks through an abstract agent model and an extensible transformation infrastructure. Currently, the tool supports three different agent frameworks of the JIAC family, and we feel confident that the approach holds for other frameworks as well as for the generation of multi-agent systems on heterogenous platforms.

1 Introduction

Over the last decade, Agent Oriented Software Engineering (AOSE) has gained attention as a suitable methodology for providing quality assurance within software development processes. However, there are at least as many agent methodologies as there are agent frameworks, and each has its own drawbacks and advantages. At present, the *DAI-Labor* has three derivatives of its agent framework *JIAC* in use, each one streamlined to a specific field of application:

- JIAC IV [8] has been developed as an agent framework for telecommunication related MAS. The system design is specified in an XML syntax which differentiates three type of agents (platforms, agents and roles) and a set of components like knowledge, services, or plan elements.
- JIAC V [5] was designed to support large agent systems in a scalable way. To this end, the successful features of JIAC IV have been rebuilt with current technologies, which provided an overall improvement in performance and maintainability. The system design is based on the Spring framework to represent the supported agent types (platforms and agents) as well as their components (services or knowledge).
- MicroJIAC [3] is JIAC's lightweight edition for devices with limited resources. The system design is done by an XML based domain model, which provides classes for both supported agent types (platforms and agents) and the framework components (e.g. services and rules), which are used to define specific functionalities or reactive behaviours.

Although the JIAC frameworks — and other agent frameworks as well — feature similarities, there are subtle differences, too. Each framework uses a different model file syntax and provides different libraries.

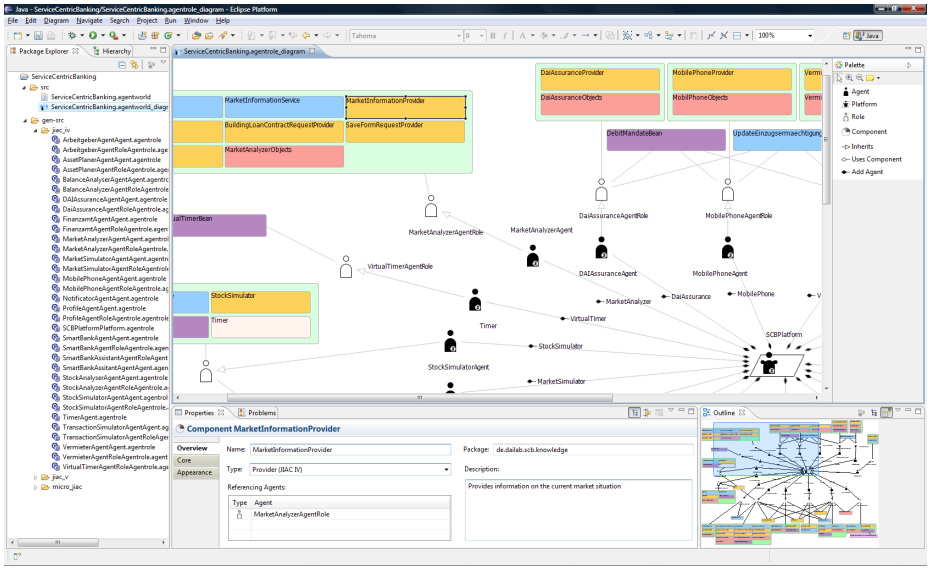

Fig. 1. Agent World Editor showing the *Service Centric Banking* scenario

An abstract conceptualisation of these native library elements enables a comprehensive, framework independent design and constitutes the main idea of the *Agent World Editor*, or *AWE*, which is described in this paper. AWE allows to design multi-agent systems (MAS) and translate the results into different agent frameworks, namely *JIAC IV*, *JIAC V*, and *MicroJIAC*. This paper is based on the diploma thesis of the first author [7].

2 Introducing the Agent World Editor

In AWE, complex multi-agent systems can be conveniently modelled in a single diagram, showing the various agent types, their components and their relationships (see Figure 1). Formal background to these diagrams is a highly generic domain model, which is capable of representing agent system instances of each JIAC derivative. After the MAS has been designed, a framework associated transformation unit can be invoked in order to produce executable agent code and stubs for potentially unimplemented components.

2.1 Generalising JIAC — The AWE Domain Model

The Domain model was designed to encompass the configuration metamodels of at least the three JIAC derivatives, but also to hold for capabilities beyond JIAC. To this end, we accounted for the diverging agent types, i.e. agents, roles and platforms, and countered the remaining challenges by more elaborate mechanisms, which we describe below.

Components: JIAC, like other agent frameworks as well, provides a set of basic agents which are capable of elementary functionality (like communication or environment awareness) and allow individual extension by appending closed and reusable components. The domain model reflects this mechanism with the `Component` class, which is used as an abstraction from the component architectures of specific frameworks.

Library Support: In order to provide support for library elements of multiple agent frameworks, AWE's domain model features the `Concept` class, which is used to provide a framework independent description of their functionality. These conceptualisations are part of AWE's base application and accessed by each installed framework extension by defining an according implementation. Although a comprehensive set of concepts is already provided, AWE's modular architecture allows for easy extension as well. Currently we support development with basic concepts like standard agency, but also provide advanced concepts like an SMS gateway, a calendar feature, and others.

Framework Openness: In order to support frameworks beyond the JIAC scope, each domain model element provides a Key-Value property mechanism. Simple aspects which are not covered by the current model (e.g. teams) can thus be defined. The decision on how these additional attributes are specified as well as their interpretation lies with the developer of the framework extension. An additional object property mechanism enables extensions in an arbitrary granularity.

2.2 Implementation

AWE has been implemented using Eclipse GMF and is based on a plug-in architecture, where each plug-in realises a distinct part of functionality. Support for each agent framework is respectively encapsulated by an individual plug-in, which is loosely coupled to the base application. The standardised structure of these extension plug-ins encourages development of custom solutions, as described in the next section. AWE provides system design in a Drag-and-Drop manner and supports the developer during this process with on-the-fly error validation, an expressive customised visual notation and mechanisms which help to accelerate and simplify recurring tasks. Furthermore, AWE inherits many useful features from GMF, such as unlimited undo and redo, auto-arrange, snap-to-grid, modelling assistance, a graphical outline, picture export, and many more.

2.3 Framework Extensions

Each extension plug-in includes the framework's components, defines a mapping from AWE's abstract concepts to a specific counterpart and defines a code generation for the respective configuration files. The code generation procedure works straight-forward: After the diagram has been validated, the *Agent World* model is exported, iterating over the several agents, roles and platforms, whereupon the several abstract concept elements are substituted with the respective framework-specific implementations provided by the plug-in. Currently, AWE provides extension plug-ins for the three frameworks of the JIAC family.

2.4 Transformation Example

In the following example, a setup consisting of a platform with two agents, a custom Component (`DetectorBean`) and a Concept element (`SMS_Gateway`) is mapped to both MicroJIAC and JIAC V. In both cases an XML file is created according to the syntax used for the respective framework. The component in both cases yields a reference to a Java class (creating a stub, if the class does not yet exist). Most interesting, however, is the mapping of the Concept element: Here, different classes are used for MicroJIAC and JIAC V, namely the library elements implementing this concept as specified in the transformation plug-ins. Figure 2 shows a schematic representation of the transformation.

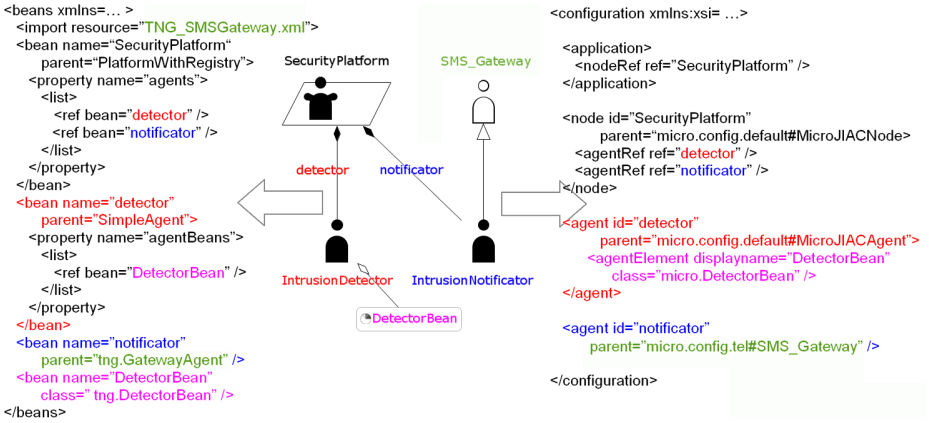

Fig. 2. Code Generation to JIAC V (left) and MicroJIAC (right)

3 Related Work

We have evaluated a number of similar tools, some of them already seeing the 3rd generation [1, 9].

The *agentTool* system [10, p. 245–259] is a visual design environment for top-down design of multi-agent systems. In its 3rd edition it is supplemented by a consistency checker, code generation, a metrics calculator and process engineering support. All aspects of an agent-oriented design can be modelled nicely, whereas the code generation feature is more or less an open issue.

The *INGENIAS Development Kit* (IDK) [4] is a visual development tool, targeting the *INGENIAS Agent Framework*. Any model element results in a skeleton that can be extended by the programmer. The IDK has been developed on the basis of an extensible plug-in architecture [4], while the main focus here lies on code generation extensions. Currently, a full translation of the design into executable *JADE* code is provided.

The *JACK Development Environment*(*JDE*) [10, p. 261–277] supports the development of *JACK* based applications. It provides for the creation and manipulation of each JACK component by means of visual engineering and encompasses several other specialised tools. MAS design is accomplished by the *JDE Design Tool* [2], which provides visual engineering on the basis of Drag-and-Drop. Code generation and execution is provided by the JDE as well. The *Compiler Utility Tool* translates the developed diagrams into Java classes and supports both, execution and debugging.

The *Component Agent Framework for domain-Experts* (*CAFnE*) Toolkit [6] provides domain experts a suitable way to easily build multi-agent systems. CAFnE operates on a framework-unspecific domain model, which allows for platform independent design. The toolkit supports visual modelling, code generation, compilation and execution of agent based applications. Currently, a complete transformation module for JACK is available, which converts the platform independent domain model to an executable agent design by using a transformation configuration and a set of transformation rules. The entire mechanism is particularly interesting for this work, since a similar feature is developed as well.

The *JIAC IV Toolipse* [9] is an IDE which facilitates MAS development within the JIAC IV framework. *Toolipse* provides visual engineering of agent applications in terms of diagrams and supports the *JIAC Methodology*. Toolipse encompasses a set of tools, each one for a specific task. Particularly interesting for this work is the *Agent Role Editor* (*ARE*), which is used during the application deployment, in order to define the MAS design. Since ARE has been applied for several years now, various improvement ideas have been considered and inspired this work. Especially its separated representation of agent roles, agents and platforms is considered a major drawback in the system design of multi-agent systems using Toolipse.

4 Conclusion

The Agent World Editor (AWE) is a tool for the visual design of multi-agent systems. It can be used for configuring the various roles, agents, their distribution on several platforms, as well as the components to be used by the agents, e.g. knowledge and capabilities. The aim of the AWE is to be framework independent and highly extensible, such that one agent world model can be used to create setups for different agent frameworks. For this purpose, we introduced the notion of the Concept element, representing an abstract capability of an agent without the need to commit to a specific framework. Upon code generation, Concept elements referenced by an agent in the design are mapped to counterparts in the targeted agent framework. Until now, we have implemented framework extension plug-ins for the JIAC family: JIAC IV, JIAC V and Micro-JIAC. For the near future we are planning to extend this scope by a respective implementation for the JACK framework. Furthermore, AWE will be extended with additional functionality – e.g. a visualisation of the interdependencies of

the several components, especially regarding communication – and it will be integrated into a larger tool suite: While AWE allows for connecting existing components to agents, the development of these components is as yet not supported. Altogether, a versatile set of tools will be combined to an IDE to provide a uniform development application for agent systems.

References

1. The agentTool III Project. http://agenttool.cis.ksu.edu/.
2. Agent Oriented Software Pty. Ltd. *JACK Intelligent Agents — Design Tool Manual*, 5.3 edition, June 2005. http://www.aosgrp.com/documentation/jack/DesignTool_Manual.pdf.
3. Tuguldur Erdene-Ochir and Marcel Patzlaff. *Programming Multi-Agent Systems*, chapter Collecting Gold: MicroJIAC Agents in Multi-Agent Programming Contest, pages 251–255. Springer Berlin/Heidelberg, 2008.
4. Ivan García-Magariño, Jorge J. Gómez-Sanz, and José R. Pérez Agüera. A Complete-Computerised Delphi Process with a Multi-agent System. In *Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, Estoril, Portugal*, pages 187–202, 2008.
5. Benjamin Hirsch, Thomas Konnerth, and Axel Heßler. Merging Agents and Services — the JIAC Agent Platform. In Rafael Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 159–185. Springer Berlin/Heidelberg, 2009. To appear.
6. Gaya Jayatilleke, John Thangarajah, Lin Padgham, and Michael Winikoff. Component Agent Framework for domain-Experts (CAFnE) Toolkit. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan*, pages 1465–1466. ACM, May 2006.
7. Marco Lützenberger. Development of a Visual Notation and Editor for Unifying the Application Engineering within the JIAC Framework Family. Diploma thesis, Technische Universität Berlin, Berlin, Germany, March 2009.
8. Ralf Sesseler. *Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten*. PhD thesis, Technische Universität Berlin, January 2002.
9. Erdene-Ochir Tuguldur, Axel Heßler, Benjamin Hirsch, and Sahin Albayrak. Toolipse: An IDE for Development of JIAC Applications. In *Proceedings of PROMAS08, Estoril, Portugal*, May 2008.
10. Gerhard Weiß and Ralf Jakob. *Agentenorientierte Softwareentwicklung — Methoden und Tools*. Xpert.press. Springer Berlin/Heidelberg, 2005.