INTERCONNECTED TOOL-ASSISTANCE FOR DEVELOPMENT OF AGENT-ORIENTED SOFTWARE SYSTEMS

Karl-Heinz Krempels, Andriy Panchenko, Janno von Stülpnagel and Christoph Terwelp
Informatik 4, Intelligent Distributed Systems Group, RWTH Aachen University, Aachen, Germany

Keywords: Agent-oriented software engineering, Tool integration, Ontology design, Problem solving method.

Abstract: The development of autonomous software systems requires the cooperation of professionals in multiple domains. This involves experts on the domain of discourse (application domain), knowledge engineers, experts in problem solving methods (PSM), as well as in the chosen deployment technology (e.g. agent technology, web services, etc.). Tool assistance for the process of developing an agent-based software systems has a direct impact on the feasibility of the planned system as well as on the acceptance of agent technology by software developers. In this paper we discuss a general approach for the process of developing agent-based software and the lack of existing tools that support the entire development process. Thereafter we describe our improvement of the development process realized by interconnected tools for ontology and PSM development, as well as as an improvement of the intergration of PSMs into a multiagent system. Finally we describe an example of an enhanced system deployment with the help of this approach.

1 INTRODUCTION

Developing agent-based software systems requires a deep analysis of the problem in order to choose a suitability software engineering method under the concrete circumstances. Agent-based solutions should be preferred mainly in cases where the application has obvious advantages in comparison to other approaches. The Agent Technology (AT) meets many requirements of problems in dynamic environments that require a distributed solution with autonomous components.

The use of AT in a distributed system is favored by FIPA[1]-compliant (FIPA, 2009) Multiagent Systems (MAS). The MAS acts as middleware, providing the abstraction for agents from the underlying operating system and facilitating the communication through the abstract Agent Communication Channel (ACC) interface in order to allow transparent use of services and interaction with other agents. Agents can be physically distributed and can work parallel on suitable problems.

The application of AT in a dynamic environment requires that agents continuously adapt to changes in their environment. This is realized by means of adaptivity and learning aptitude of autonomous agents. However, it should be noted that these features are explicitly realized by developers as they are not present in agents per se.

The development of software systems with help of agent technology is a process consisting of phases ranging from analyzing the domain, through designing and implementing specific ontologies and PSMs, up to implementing the agent. The deployment method can be based either on one specific tool facilitating all these phases or on suitable tools for facilitating single phases of the process (Krempels, 2008). The later requires usage of completely independent tools which are neither complementary nor interoperable.

The advantage of the former approach is the homogeneity of tool assistance through the whole development process. This facilitates consistent representation and usability of the model and the outcomes of the deployment phases. A disadvantage is a mostly longsome skill adaption training for highly complex tools. The advantage of the latter approach (using the generic development method) is the possibility of using well-known, established tools that are usually known to the developers and very well suited for sin-

[1]Foundation for Intelligent Physical Agents

Krempels K., Panchenko A., von Stülpnagel J. and Terwelp C. (2009).
INTERCONNECTED TOOL-ASSISTANCE FOR DEVELOPMENT OF AGENT-ORIENTED SOFTWARE SYSTEMS.
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, pages 308-314
DOI: 10.5220/0002329903080314

gle phases.

The lack of interfaces for exchanging data and outcomes among the different phases, however, causes problems with its applicability. The first challenge developers are faced with is the transformation of data representation between the different phases. Due to the fact that different, often incompatible models and knowledge representation formats are used, it is required to transform the data while losing as little information as possible. Usually the developers end up with an iterative process of analysis, modeling and implementing the problem scenario with the help of different tools until the desired result is obtained (Krempels et al., 2003).

In this paper we discuss a general approach for an agent-based software development process and the lack of tools supporting the entire development process. Thereafter, we focus on our contribution to the development process assistance in the form of tools for linking ontology to the development of problem-solving methods, as well as an integration of the PSM into a MAS. Finally, we provide a proof for the concept implementation following our approach (Krempels and Panchenko, 2007) (Kirn et al., 2003) (Krempels and Panchenko, 2006). This is in fact the development of a planning system with the help of AT.

2 GENERIC TOOLS' ASSISTANCE

There exists a variety of application domains for software systems. Different application domains, however, raise quite diverse requirements on the model representation. This leads to the availability of several domain-specific modeling tools that make use of different representation languages. Therewith, the export of a model in a form that is suitable for processing it in another tool is often very problematic. Agent-based software development usually consists of four phases (Krempels, 2008): domain analysis, ontology design, PSM implementation, as well as integration into the implementation of the agent or agent society. In the following we discuss these phases as well as current assistance of the modeling process in detail.

2.1 Domain Analysis

The goal on this level is to provide a domain or task description in form of a model. The design of such a model is based either on an expert interview, process flow analysis, or statistical analysis. Usually domain-specific modeling tools are applied in order to facilitate the process. In domain modeling one of the most acceptable and widely popular tools are ARIS-Toolset[2] and Microsoft Visio[3]. The former one supports modeling, optimization as well as simulation of processes and provides a possibility to adapt an optimized process for the real application. The modeling language in use is called Event-driven Process Chain (EPC). It provides a possibility to export/import modeling data to/from the Unified Modeling Language (UML). The outcome of the modeling is a process model that inherits background information. Visio is a simple modeling tool that provides no support for analyzing the resulting models by any means. Besides above mentioned modeling languages EPC and UML, also additional formats can be used. The latter, however, provides only a graphical layer without any semantic consideration. Visio is mostly used for fast deployment of less complex models. The outcome is a basic graphical representation of the model. The domain modeling is depicted in Fig. 1 in the first column.

2.2 Ontology Design

The goal of the second step in the design process is to develop an ontology. An ontology is an explicit specification of a conceptualization (Becker et al., 2003a). It offers the ability to share and reuse knowledge about a common universe of discourse (Kirn et al., 2003). The design and deployment of ontologies is based on the process models from the domain level. There exists a number of tools that support the ontology design process, e.g. Protégé[4], OilEd[5], WebOnto[6]. The most suitable and widely used tool in MAS environment is Protégé. It supports a frame-based ontology design and it is possible to create instances from the deployed concepts in order to feed them with the acquired domain knowledge.

For this purposes it is possible to make use of automatically generated graphical forms that can be adapted to the user's needs in order to simplify knowledge acquirement. The base functionality of Protégé may be extended with the help of dynamically loadable libraries (plugins). At the moment there exist plugins with interfaces to databases (e.g. through the Java Database Connectivity (JDBC) to SQL), visualization tools (e.g. in UML), expert systems (e.g. C Language Integrated Production System (CLIPS)) as well as for an export of ontologies into differ-

[2]IDS Scheer GmbH – http://www.ids-scheer.de/

[3]Microsoft Inc. – http://www.microsoft.com/

[4]The Protégé Ontology Editor and Knowledge Acquisition System – http://protege.stanford.edu/

[5]OilEd – http://oiled.man.ac.uk/

[6]WebOnto – http://kmi.open.ac.uk/projects/webonto/

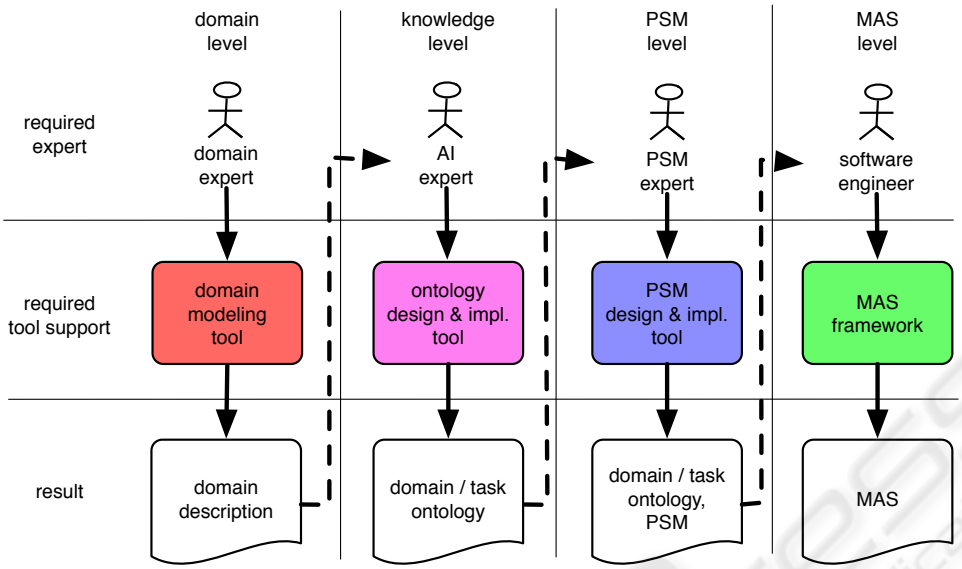

Figure 1: Tools' Assistance in the Development Process.

ent knowledge-representation formats (e.g. RDF and XML) or as Java Beans (BeanGenerator, 2007). The outcome at this stage is an ontology in one of the above mentioned representation formats and, if the knowledge acquirement pursued, a knowledge base with instances of the modeled concepts. These can be directly used with the ontology described by the scenario for the analysis of PSMs. The ontology modeling is depicted in Fig. 1 in the second column.

2.3 PSM Implementation

Problem solving methods describe a solving procedure for a concrete task. This is usually a generic procedure. Tool assistance for the PSM deployment depends on the degree of formalization as well as on the chosen implementation language. Examples are: rule-based systems (RBS), declarative and procedural languages. The latter, however, do not provide support for ontologies. Especially because of the autonomous adaptive behavior which is demanded from agents, it is very challenging to use a RBS on this stage. The implementation can be supported by making use of the PSM libraries that include several solution methodologies for the standard class of known problems, e.g. timetabling, ressource coordination, scheduling, planning, etc. PSMs are usually developed by experts on algorithms and artificial intelligence (AI). The process of PSM development is depicted in Figure 1 in the third column.

2.4 Agent Development

On this level the domain description in the form of an ontology together with the problem-solving methods is integrated into an agent. Additionally, the interaction capabilities have to be implemented in this phase in order to be able to communicate with other agents and act autonomously. There exists a variety of tools for the deployment of FIPA-compliant MAS. Many of them partly provide own development methods for agents and agent societies. At this stage the choice of a suitable platform for agent-based software development has to be made. One of the most popular FIPA-compliant MAS is JADE (JADE, 2009).
The process of software development as described in this section is a typical and widely adopted way of agent-based software engineering. It is supported by generic, well established tools. This process, however, is characterized by a conceptual disadvantage because of the problematic usability of outcomes from the individual phases of the process.

3 TOWARDS INTERCONNECTED DEVELOPMENT

The poor interconnectivity between the phases in the state-of-the-art development process described above causes problems in data exchange between them. This leads to information loss during the transformation of models from one phase into the next one. The problem can be solved either by integrating established

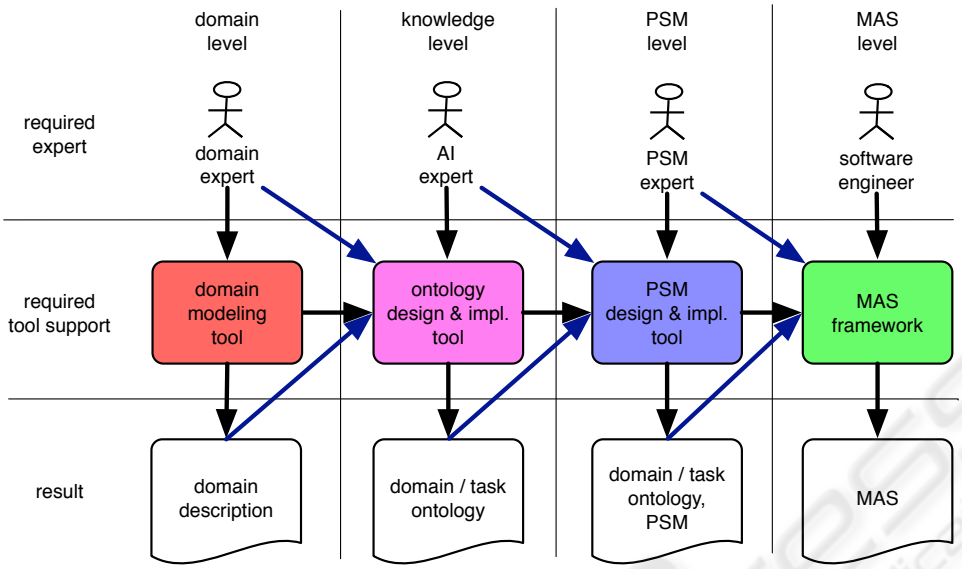

Figure 2: Integrated Tools' Assistance in Development Process.

tools that facilitate single phases or by developing a single tool that would support all the phases of the process. We follow the first approach: we propose and implement interfaces that support integration of the results from one tool into the one used in the subsequent phase. Therewith the information flow on the tool level follows the path as depicted in Figure 2.

To this end we propose an improved assistance for the development of agent-oriented software systems which is twofold. The first improvement interlinks ontology with the development of problem solving methods (JamochaAgent Tab). The second one is the integration of PSM into MAS (JamocaAgent). There are two possible approaches to consider here. The first one is to modify the commonly used tools to support the same data formats. The second one is to develop an interconnection component that supports conversion between different formats. We followed the second approach.

3.1 JamochaAgent Tab

Jamocha (Jamocha Community, 2009) is a rule engine and scripting environment written entirely in Java. Jamocha was originally inspired by CLIPS expert system shell, but has grown into a complete and distinct Java-influenced environment of its own. Written in the Java programming language, Jamocha offers easy integration into other Java-based software. Because most of the MAS are Java-based, Jamocha was preferred over other rule engines written in other languages.

In order to facilitate the export of ontologies from Protégé into a template-based manner which is understood by Jamocha, the *JamochaAgent Tab* was developed. It is realized as a plug-in for Protégé. The *JamochaAgent Tab* exports ontologies in the following forms:

- templates with restricting rules;
- concrete instances;
- definitions of agent actions.

Templates are representations of class concepts from Protégé in the CLIPS language. Abstract classes are not exported as templates since they can not have direct instances. However, each concrete subclass inherits all slots of its parents. There is no possibility to model class hierarchy in Jamocha directly. That is why it is modeled artificially, adding a new multislot *is-a* to each subclass, where all parents are listed. Doing so, it is not always possible to reconstruct the right classes hierarchy (since the ancestor slot is not ordered), especially when abstract classes are used. The latter, however, is not a problem for a RBS since the main use of hierarchy here is to check whether one fact is an instance of another one. Furthermore, rules are used in order to check existing slot constraints (like min- max values, symbols, etc.). If an inserted fact does not follow the constraints, the corresponding rule will fire and the fact causing the violation will be retracted. An example of the *date* class mapping into Jamocha is shown below:

```
(deftemplate date
  (slot day (type INTEGER))
```

```
  (slot month (type INTEGER))
  (slot year (type INTEGER))
  (multislot is-a
   (default (create$ TemporalConcept))))
```

It can be seen, that *date* is a subclass of a *Temporal Concept*. For the purpose of having a possibility to extract names of all the parent classes by a child name and vice versa, there is also another kind of facts generated:

```
(assert (isA
 (parent Preference)
 (child CoworkerPreference))
)
```

Further, one of the rules used in order to check the maximum day value of the date is captured. Whenever a fact of type *date* with a slot *day* is added or changed and the value of this slot exceeds 365, the rule fires. As a result the *retract* function is called, which deletes the fact that violates the constraint:

```
(defrule constraint.date.day.max
  ?fact <- (date (day ?slotvalue))
  (test (> ?slotvalue 365))
 =>
  (retract ?fact)
)
```

An instance of the template *date* looks like this:

```
(assert
  (date
     (day 15)
     (month 3)
     (year 2004)
))
```

AgentAction is a special class, which is used for communication purposes with other FIPA-compliant agents. Requesting another agent to perform some action requires the specification of an action expression that consists of an agent id (the id of the agent requested to perform the action) and an instance of an *AgentAction* (a formal description of the requested action, e.g. a function name). Since there is no equivalent language element available in the CLIPS language the *AgentAction* is mapped to a function call in CLIPS. So, whenever an agent is requested to perform an action, the corresponding CLIPS function with the same name as the *AgentAction* KB is applied to the the agent's KB.

All the subclasses of the *AgentAction* concept are mapped into those functions. An example of *AgentAction addTask* that simply adds a received task to the fact-base is captured below:

```
(deftemplate addTask
  (slot aTask)
  (type Task)
)

(deffunction addTask (?jat_arg_fact)
  (bind ?aTask
    (fact-slot-value ?jat_arg_fact aTask))
   ...
  (retract ?jat_arg_fact)
)
```

3.2 JamochaAgent

One of the fundamental reasons for using agents is making use of their ability to reason, communicate, and act autonomously. Communication is facilitated by using a MAS, such as JADE – a FIPA-compliant multi-agent system. The ability to reason and act rationally is usually programmed in an expert system, which itself is based on a RBS.

JamochaAgent (Krempels et al., 2009) is a JADE-agent that has an embedded Jamocha engine and a user-friendly interface, allowing it to inspect and to modify the knowledge base, facts, rules, and functions at runtime. The architecture of the *JamochaAgent* is depicted in Fig. 3. This agent inherits FIPA-compliant communication capabilities from the agent component provided by the JADE framework and serves therein only as a communication interface to other agents. The interaction among agents consisting of consecutive speech acts is described by FIPA interaction protocols which are mapped onto the RBS to allow for interaction protocol adaption at runtime. Incoming speech acts are consumed and outgoing speech acts are produced by the RBS in rule-based manner. *JamochaAgent* agent provides support of most commonly used FIPA interaction protocols (Request, Query, Contract Net) within *Jamocha*.

4 APPLICATION EXAMPLE

We have validated our approach while developing a scheduling system for medical treatments. Planning medical treatments and scheduling surgical operations are substantial elements of the hospital management. Operations theater scheduling deals with assigning limited hospital resources (rooms, doctors, nurses, etc.) to jobs (patient treatments, surgery, etc.) over time in order to perform tasks according to their needs and priorities, and to optimize the usage of hospital resources (Krempels and Panchenko, 2006).

The analyzed application scenario was modeled in Protégé. The outcome is the task which formally defines objects in the scenario and relations among

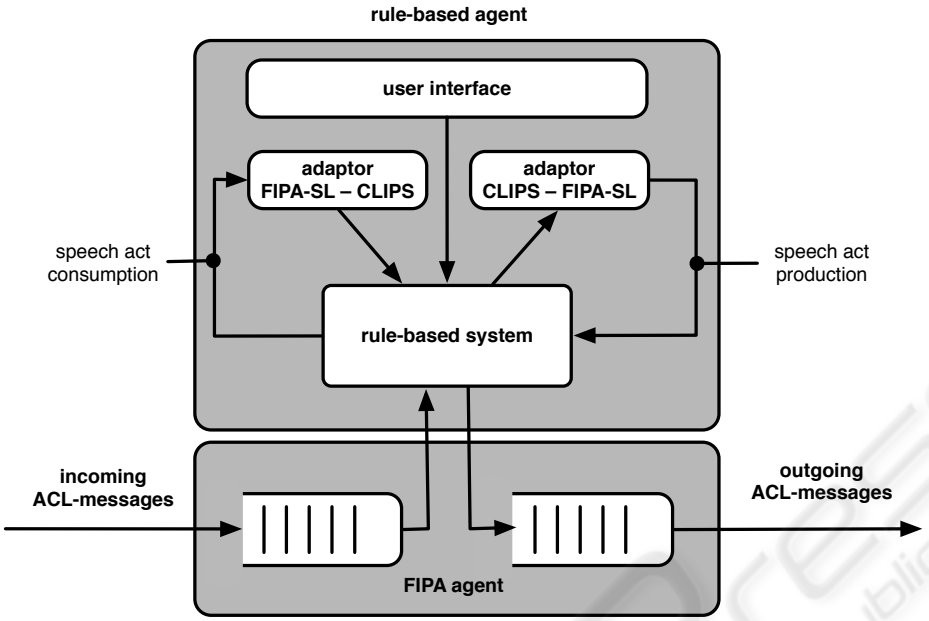

Figure 3: *JamochaAgent* Architecture.

them. The domain ontology OntHoS (Becker et al., 2003b) was used as a reference to develop the own task ontology. The concepts together with their instances are exported as facts and rules into the RBS *Jamocha*. The RBS is used as an environment to implement PSM in. These are scheduling heuristics and conflict solving mechanisms.

With the help of the *JamochaAgent* the developed heuristics are applied into the MAS JADE, which is used as a middleware. Further, all the agents (*SchedulingAgent* and *WardAgent*) are started and the scheduling process is initiated by the *SchedulerAgent*. The user interfaces for interacting with the planner as well as the generated subplans (based on ontological constraints) are provided by this agent. New scheduling tasks are added to the system by a ward's representatives with the help of *WardAgent*'s.

The deployed *SchedulingAgent* is based on *JamochaAgent* because of its suitable integration into the optimized development process. This means that the PSM and the behavior of the *JamochaAgent* are written in a higher programming language which does not require source code compilation. Task ontologies and PSM are loaded at runtime as well as the new fact base reflecting a possible change in the considered scenario.

5 CONCLUSIONS

The process of developing agent-based software suffers from the lack of tools supporting the entire development process. Thus, different steps of the process are pursued by completely different tools having incompatible model descriptions and knowledge representation formats. This results in a complicated deployment process which is done in an iterative way.

We described an improvement of the development process realized by interconnected tools for ontology and PSM development, as well as as an improvement of the intergration of PSMs into a MAS.Finally we provided an example of an enhanced system deployment with the help of this approach.

Single phases of the development process are pursued, depending on skills, by experts from different domains. Successful and effective cooperation requires their consensus regarding the concepts and solution methods. However, based on our experience, this causes greater problems than the partial lack of tool assistance or the limitations of the available tools.

ACKNOWLEDGEMENTS

This research was funded in part by the DFG Cluster of Excellence on Ultra-high Speed Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.

REFERENCES

BeanGenerator (2007). *Beangenerator Plug-in*. http://acklin.nl/beangenerator/.

Becker, M., Heine, C., Herrler, R., and Krempels, K.-H. (2003a). OntHoS - an Ontology for Hospital Scenarios. In *First International Workshop on Agent Applications in Health Care*, Barcelona, Spain.

Becker, M., Heine, C., Herrler, R., and Krempels, K.-H. (2003b). OntHoS - an Ontology for Hospital Scenarios. *First International Workshop on Agent Applications in Health Care, Barcelona, Spain*.

FIPA (2009). *FIPA - Foundation for Intelligent Physican Agents, Home Page*. http://www.fipa.org/.

JADE (2009). *JADE Home Page*. http://jade.cselt.it/.

Jamocha Community (2009). *Jamocha Home Page*. http://www.jamocha.org/.

Kirn, S., Heine, C., Herrler, R., and Krempels, K.-H. (2003). Agent.hospital - agent-based open framework for clinical applications. In *WETICE*, pages 36–41.

Krempels, K., Christoph, U., and Wilden, A. (2009). Jamocha – a Rule-based Programmable Agent. In *Proceedings of the International Conference on Agents and Artificial Intelligence*, Porto, Portugal.

Krempels, K.-H. (2008). *Agentenbasierte Ablaufplanung*. PhD thesis, RWTH Aachen University, RWTH Aachen University, Germany.

Krempels, K.-H., Nimis, J., Braubach, L., Pokahr, A., Herrler, R., and Scholz, T. (2003). Entwicklung intelligenter Multi-Multiagentensysteme - Werkzeugunterstützung, Lösungen und offene Fragen. In Dittrich, K., König, W., Oberweis, A., Rannenberg, K., and Wahlster, W., editors, *Informatik 2003 - 33. Jahrestagung der GI*, pages 31–46. Gesellschaft für Informatik e.V., Köllen Druck+Verlag GmbH Bonn.

Krempels, K.-H. and Panchenko, A. (2006). An Approach for Automated Surgery Scheduling. In Burke, E. and Rudová, H., editors, *PATAT 2006 - Proceedings of The 6th International Conference on the Practice and Theory of Automated Timetabling*, Brno, Czech Republic. Masaryk University.

Krempels, K.-H. and Panchenko, A. (2007). KR-driven Development Process Integration. In Vendetti, J., Hopper, T., and Tudorache, T., editors, *Proc. of 10th Intl. Protégé Conference*, Budapest, Hungary.