JIAC V — A MAS Framework for Industrial Applications

(Extended Abstract)

Marco Lützenberger Tobias Küster Thomas Konnerth Alexander Thiele Nils Masuch
Axel Heßler Jan Keiser Michael Burkhardt Silvan Kaiser Sahin Albayrak
DAI-Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany
firstname.lastname@dai-labor.de

ABSTRACT

Agent-oriented research and technologies have produced a number of relevant and valuable results. However, industrial stakeholders are still reluctant to adopt agent technologies for their products. Modern industrial projects put a set of requirements on software frameworks, which are often neglected by contemporary agent frameworks. In an attempt to close this gap, we present the JIAC V agent framework.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Distributed Artificial Intelligence—*Multiagent systems*

Keywords

agent framework; mas development; industrial adoption

1. THE JIAC V FRAMEWORK

The 5th version of the *Java Intelligent Agent Componentware*, or *JIAC*, is a multiagent system development framework and runtime environment, that has been applied in a number of research- and industrial projects [1, 3, 5].

One of the core aspects of JIAC is the integration of agents with the service oriented architecture paradigm (SOA) [2]. A reliable discovery and messaging infrastructure ensures, that JIAC agents can be distributed transparently over a network, or even beyond network boundaries.

An agent-platform comprises one or more 'agent nodes' which are physically distributed and provide the runtime environment for JIAC agents. New agents, services, as well as further agent nodes can be deployed at runtime. Agents interact with each other by means of service invocation, by sending messages to individual agents or multicast channels, and by complex interaction protocols. Each individual agent's knowledge is stored in a tuple-space based memory. Finally, JIAC agents can be remotely monitored and controlled at runtime via the *Java Management Extension Standard* (JMX).

Each agent contains a number of default components, such as an execution-cycle, a local memory and the communication adaptors. The agents' behaviours and capabilities are implemented in a number of so-called *AgentBeans*. AgentBeans support very flexible activation schemes: A bean may be executed at regular intervals or according to a life-cycle change, such as *initialised*, or *started*. Further, AgentBeans can attach *observers* to the agent's memory, being called for instance each time the agent receives a message or updates its world model.

AgentBeans also provide *action* methods, which are exposed to the directory and invoked either within the agent or by other agents. By using these mechanisms, it is possible to define all of the agents' capabilities and behaviours. The entire multiagent system is then defined by one or more *Spring*[1] configuration files.

1.1 Default and Extension AgentBeans

JIAC agents contain a set of default AgentBeans that constitute core functionalities of an agent, as well as a number of individual AgentBeans, that are implemented as described above. One such AgentBean each JIAC agent is equipped with by default is the *Communication Bean*. This component manages the inter-agent service communication and also allows agents to exchange messages with other agents or groups of agents in the network, addressing individual agents or multi-casting to message channels. The messages are not restricted to FIPA messages but can have any data as payload.

Complementary to the AgentBeans, there are *NodeBeans*, adding functionality to a node as a whole. Each agent node is by default equipped with a *Directory Node Bean*, listing the actions of the different agents, and a *Message Broker Node Bean*, being the counterpart to the agent's communication bean and allowing them to transparently send messages from node to node using *ActiveMQ*[2].

Other commonly used AgentBeans and NodeBeans can be added to a multiagent system by simply adding the bean to the agent's configuration. For the composition of services, JIAC includes an *Interpreter* AgentBean for the execution of the high-level service-oriented scripting language *JADL++* [2]. Reactive behaviour of agents can be enabled with a *Drools*[3] rule engine that can be synchronised with the agents' memory.

Extensions to the capabilities of nodes and agents include a *Migration* NodeBean, that enables strong agent migration

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

[1]Spring: `http://www.springsource.org/`
[2]ActiveMQ: `http://activemq.apache.org/`
[3]JBoss Drools: `http://www.jboss.org/drools/`

between agent nodes, a *Persistence* NodeBean that saves the node configuration and allows for restarting the node later on, and NodeBeans for *Load Measurement* and *Load Balancing* that provide cross-node load information and distribute agents over nodes at start- and runtime. In order to support application development, JIAC also provides generic functionalities such as AgentBeans for *User Management*, *Human Agent Interfaces*, a *Webserver* NodeBean running an embedded Jetty-server, and a *Web Service Gateway* AgentBean that exposes JIAC actions as web services and vice versa. Last but not least, the *OSGi Gateway* allows JIAC nodes to be executed within an OSGi framework and to access other OSGi services.

1.2 Development Methods and Tools

To improve the development efficiency, JIAC provides a set of additional tools, all of which are integrated directly into the Eclipse IDE. To start with, the *JIAC Project Wizard* helps with creating new JIAC projects by generating a uniform project structure, including a readily configured Maven `pom.xml` file, as well as a starter class for running the new JIAC application. Furthermore, several Eclipse views provide information about currently running JIAC applications on the network and the agents and services they contain, as well as the possibility to start and interact with newly created agents and services. Complementary to those basic development tools and utilities, JIAC agents can be modelled using two high-level graphical editors. The *Visual Service Design Tool* [4] allows for the modelling of workflows and interactions of individual agents by means of a series of BPMN diagrams. Based on these diagrams, executable JIAC AgentBeans or JADL++ services can be generated. The *Agent World Editor* (AWE) [4] can be used to create diagrams, showing the different agents and agent nodes in a distributed system and the individual services they provide. From these visual models, the AWE can generate the corresponding Spring configuration files and JIAC AgentBean stubs. Finally, the running multiagent system can be monitored and manipulated using the *ASGARD* runtime monitor [6], providing an intuitive, three-dimensional view of all agent nodes and agents running in the local network as well as the communication between them.

1.3 Application Frameworks

JIAC was also used for a number of frameworks, which we present in this section. While these frameworks have usually been developed for specific domains, they are generic enough so they can easily be adapted and reused in other contexts. *NeSSi*^{2}, the Network Security Simulator [1], was originally developed for checking the security of computer networks, but is also capable of simulating other networks, e.g. power grid, or ICT networks. Also, a generic *Optimisation* framework can be used to run an optimisation, distributed among several agents [3]. It can be used with different optimisation algorithms and problem domains. JIAC was also used for the development of a *Driver Simulation* framework [5]. Using a mental model, the framework is able to simulate a special form of human traffic behaviour, namely strategic compensatory behaviour. Finally, the *Eclipse Plugin Node* is an Eclipse plugin running a JIAC node inside of the developer's IDE. This allows e.g. to use JIAC's discovery mechanisms to display available nodes and agents in a view and to deploy or invoke actions on those nodes.

2. FINAL REMARKS

We certainly recognise that there are many multiagent frameworks available – each one, with a focus on particular system characteristics. Yet, as of today, the agent community was not able to convince industrial players to adopt their ideas. JIAC was never intended to include the cutting edge of agent research but to constitute a robust, reliable, homogeneous and well-documented framework for the development of agent-based software applications. It was also our intention to equip JIAC with features which are generally required for extensive industrial appliances. Today, the JIAC framework has proven its applicability. Common requirements were integrated as core functionalities, while the additional features can be easily implemented, due to the openness and the extensibility of the framework. We do not consider JIAC to be an ultimate solution for the discrepancy between agent research and the applying industry. Yet, given the fact that JIAC was originally streamlined towards industrial projects and ease of use, it is our opinion that JIAC has the potential to provide new incentives for industrial stakeholders and users which are not all too familiar with the agent paradigm, to adopt multiagent technology.

3. REFERENCES

[1] D. Grunewald, M. Lützenberger, J. Chinnow, R. Bye, K. Bsufka, and S. Albayrak. Agent-based network security simulation (demonstration). In K. Tumer, P. Yolum, L. Sonenberg, and P. Stone, editors, *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan*, pages 1325–1326, May 2011.

[2] B. Hirsch, T. Konnerth, and A. Heßler. Merging agents and services – the JIAC agent platform. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 159–185. Springer, 2009.

[3] T. Küster, M. Lützenberger, and D. Freund. Distributed optimization of energy costs in manufacturing using multi-agent system technology. In *Proceedings of the 2nd International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 53–59, Maho Bay, Sint Maarten, March 2012.

[4] T. Küster, M. Lützenberger, A. Heßler, and B. Hirsch. Integrating process modelling into multi-agent system engineering. *Multiagent and Grid Systems*, 8(1):105–124, January 2012.

[5] M. Lützenberger, S. Ahrndt, B. Hirsch, N. Masuch, A. Heßler, and S. Albayrak. Reconsider your strategy—An agent-based conceptualisation of compensatory driver behaviour. In *Proceedings of the 15th Intelligent Transportation Systems Conference, Anchorage, AK, USA*, pages 340–346, September 2012.

[6] J. Tonn and S. Kaiser. Asgard—A graphical monitoring tool for distributed agent infrastructures. In Y. Demazeau, F. Dignum, J. M. Corchado, and J. B. Pérez, editors, *Advances in Practical Applications of Agents and Multiagent Systems*, volume 70 of *Advances in Intelligent and Soft Computing*, pages 163–173. Springer Berlin Heidelberg, 2010.