Agent Programming in Practise — Experiences with the JIAC IV Agent Framework

Benjamin Hirsch
Benjamin.Hirsch@dai-labor.de

Stefan Fricke
Stefan.Fricke@dai-labor.de

Olaf Kroll-Peters
Olaf.Kroll-Peters@dai-labor.de

Thomas Konnerth
Thomas.Konnerth@dai-labor.de

DAI Labor
Technische Universität Berlin

ABSTRACT

This paper describes the agent framework JIAC IV, and the various projects it has been applied in. The projects were industry-funded as well as research-oriented. Our aim is to convey the particular requirements that followed from this link, and the consequences for the further development of the framework. We describe the projects and the impact that JIAC IV had on them, and conclude with an analysis and the current state of affairs.

1. INTRODUCTION

Agent technology has been around for a number of years now, and the still growing number and size of the various workshops and conferences in the field show that the importance of agents and related technologies is high. It is however also the case that even though a large number of researchers and practitioners work on agents, a much smaller number has managed to breach the wall between academia and industry. There are commercial frameworks available, and companies advertise the fact that they employ agent technology, but in general it can be said that the impact that agent technology has had on the market is less that it could have been. Reasons given for this failure — or even whether or not to call it a failure — vary wildly. We maintain that one of the reasons is the missing interaction between agent researchers and industry, sometimes leading to solutions that are not applicable or usable in "real" applications.

JIAC IV (Java Intelligent Agent Compontentware) is an agentframework that has initially been financed by Deutsche Telekom, and was from the very beginning designed to cope with industry requirements. In a number of projects of different domains, the framework has been adapted and further refined.

This paper gives an overview over the JIAC IV agent framework, with a high level view of the features (Section 2), and proceeds to describe various projects that have been implemented using JIAC IV (Section 3). The paper concludes with a discussion of the experience made and the lessons learned during those projects (Section 4).

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

Not for citation

2. JIAC IV

JIAC IV has first been developed in 1998 [7] and has seen a number of revisions since. Its focus was initially on supporting telecommunication applications, but it quickly got used in areas as different as information retrieval, telematic services, and personal information.

The JIAC IV agent framework supports the development of multi-agent systems (MAS) using BDI agents on FIPA [15, 16] compliant platforms. JIAC IV has been implemented using the Java programming language. Two building blocks constitute the basic agent architecture: the JIAC component system [25] and the JIAC Agent Description Language (JADL) [20]. The basic architecture of a JIAC-based application is summarised in the JIAC MAS meta-model, which is shown in Figure 1.

Based on this core architecture, the JIAC-framework also includes a list of enhanced features that cover the management and security aspects as well as the software lifecycle, i.e. engineering, deployment and runtime. All of these will be discussed in the following sections:

2.1 Agent and Service-model

As can be seen in the JIAC MAS meta-model (Figure 1), JIAC has explicit notions of goal, rule, plan, service, and protocol. Each JIAC agent constitutes it's own component system and is able to accommodate any number of different components that implement either the agents' behaviour or the agents' capabilities. While some of these components are implemented using Java-beans, our notion of components also covers concepts like ontologies or scripts which are written in the language JADL [20].

2.1.1 Knowledge

The central anchor for this component based approach is the agents factbase, which stores the agents world model and is accessible from all components. The knowledge stored in this factbase is represented by ontologies that describe the data and its relations. While these ontologies described in JADL are less expressive than OWL-lite, they can be mapped to a subset of OWL [22].

Furthermore, as JIAC-agents are intended to operate in open and dynamic environments, their knowledge representation is based on 3-valued logic [19] and therefore allow to reason with uncertainty. Consequently, JIAC-agents operate with an open world model.

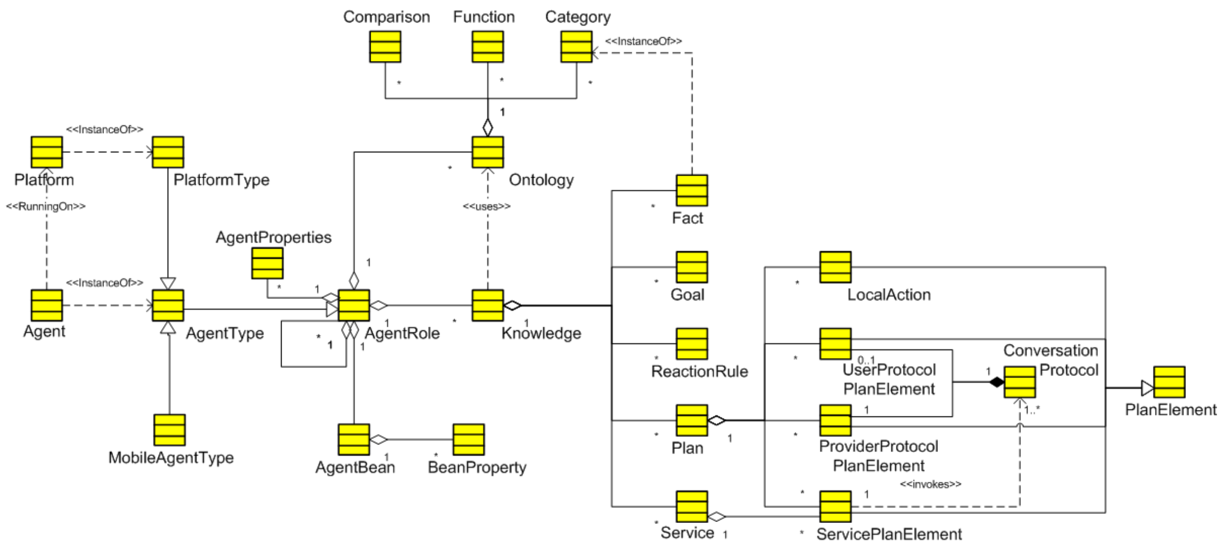

Figure 1: JIAC IV MAS meta-model

2.1.2 *Acting*

Based on this knowledge representation and its factbase, each agent employs a version of a BDI-cycle [8] in which its state-goals are evaluated against the factbase and appropriate capabilities are selected and triggered as necessary. These capabilities are called plan-elements and come in different versions. They can be local actions as well as services and they can be implemented by either a JavaBean or in JADL. Furthermore, plan-elements are allowed to invoke other plan-elements. However, as the system transparently looks up services from other agents if there is no local plan-element that can fulfil the goal, the difference between these plan-elements is more a matter of the preferred implementation language and has no effects on the runtime behaviour. Additionally, JIAC-agents can also deal with simple rules that provide a quick reaction to changes in the environment [20].

2.1.3 *Semantic service selection*

To support the dynamic and flexible selection of plan-elements, the action- and service-descriptions all contain semantic descriptions of preconditions and effects based on the JADL-ontologies. This allows the agent to verify both, whether an action is applicable and whether it reaches desired effect. Furthermore, it enables a semantic service matching, so that available services may be selected dynamically, without prior adaptation from the developer.

As an enhancement to this action selection and the BDI-cycle, JIAC agents support planning from first principles — using a UCPOP algorithm [24] they can, if no appropriate plan-element can be found, try and build a chain of services that will lead them to the desired state of affairs. Note that this goes further than just providing a number of pre-defined plans that the agent can choose from. The agent's execution will first try and find plans or services that match his goal, and only if it does not find any it will start planning.

2.1.4 *Communication*

As we have already mentioned, JIAC employs service-based interaction. More specifically, while JIAC uses FIPA compliant speech acts for communication, the communication between two JIAC-agents is always handled via service calls. Every service call is wrapped in a so called meta-protocol (which is essentially a modified *FIPA Request* protocol), which automatically takes care of security requirements, error handling, data exchange and negotiation protocols between agents. The actual exchange of messages between agents during one service session however is not restricted, as long as both agents can agree on a common protocol.

While service provision always occurs between two agents, the meta protocol also allows to precede the actual provision with a provider selection, where the service user can request a service from many agents and use any suitable protocol such as for example the contract net protocol [26] to select the most suitable provider. It can here use the semantic service description that each service has, including pre-condition, effect, and QoS information.

2.2 Infrastructure - Management and Security

Based on the core features we explained in the previous section, JIAC employs a list of advanced infrastructure functionalities which have proven differently useful in application development:

2.2.1 *Dynamic Components and Mobility*

One of the signature features of JIAC IV is its ability to exchange components during run-time. As stated earlier, components may be actions, services, rules, ontologies, but also so-called agent beans, i.e. Java components.

Furthermore, JIAC agents support strong mobility, meaning that they can be moved between hosts during run-time without having to stop execution — this presupposes however that the agents does not need software or services such as databases that are only available at certain hosts.

These two ability of the framework allows the quick adaptation of systems and applications, as the agents can automatically find and use new components, or can independently move to other hosts to e.g. reduce communication.

2.2.2 Security

While the dynamicity that follows from the component exchange and the strong mobility of agents can be quite powerful, it is also a potential security threat in real world systems. Therefore, JIAC contains elaborate security mechanisms on all levels. For example, it provides a public key infrastructure interface to authenticate agents, and supports encrypted communication between agents [9].

The Framework has been certified by the German Federal Office for Information Security (BSI) following the Common Criteria Level 3 [10]. This makes it worldwide the only agent framework to be security certified.

2.2.3 Accounting Features

As a further extension to the service mechanism, JIAC IV provides a management interface through which complex accounting and tariff schemes can be implemented [18]. This is important when agents move into the mainstream and offer services to other agents, and want to provide intelligent and dynamic billing methods. The accounting features have been tested in the course of the project Berlintainment [28].

2.3 Tools and Methodology

One last, but nevertheless important aspect of JIAC is its iterative methodology (as shown in Figure 2) that supports large projects by not only defining analysis and design but also including methods to deal with customers and team interaction. The methodology has been applied and refined in numerous projects that we have done at the DAI Labor with industrial and research partners.

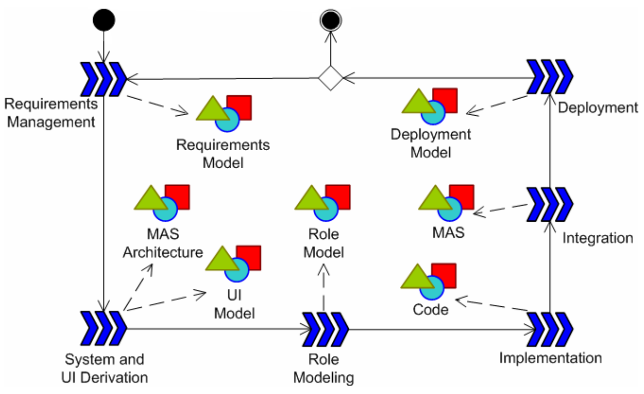

Figure 2: JIAC methodology - iterative and incremental process model in SPEM notation.

Supplementary with this methodology, JIAC comes with its own IDE which is implemented as Eclipse[1] plugin [27]. It provides textual and graphical editors for all parts of agent programming, including an ontology editor, an agent role editor, a knowledge builder, a repository for agents, and more. The IDE has extensive help functionality and has a built in tutorial to help the user get acquainted with JIAC and JADL.

3. PROJECTS

During last ten years we have used JIAC in many research projects in different domains such as personalised information services, security, healthcare, entertainment, among others. In the following, we describe some of these projects and the experiences we made.

3.1 URLAUB

The goal of the project URLAUB (German for holiday), running from September 2001 until August 2002, was a personalised system for for holiday planning. Unlike traditional booking systems, URLAUB created recommendations based on user preferences for climate, lodging, activities, and so on. These preferences were stored in a profile which was then used for providing future recommendations as well as for comparisons with other users' profiles. An agent-based approach was taken and combined with advanced filtering techniques in order to integrate a number of different travel agencies and related travel information sources, such as climate data, politics, and cultural or sports characteristics. Additionally, a device independent user interface, adaptive user management, the ability to pre-book, as well as feedback mechanisms set the system apart from standard travel booking sites.

3.1.1 Setup and Methodology

The task-oriented JIAC methodology was applied to the analysis and design, giving rise to seven task areas with a total of about 90 identified tasks, comprising, among others, the access and monitoring of remote sources, the managing of user profiles, user interfaces and devices, the processing of travel requests, information filtering as well as notification tasks. 17 roles were identified and subsequently mapped onto agent types. Each agent type combined a number of related tasks, identified through analysis of interaction between tasks. Non-functional requirements such as availability and reaction time was also taken into consideration. A total of 65 services where identified and assigned to 13 agent types. During deployment, several instances of each agent type were created in order to avoid bottlenecks, and critical functionality was distributed over several platforms running on different servers. In the first release, each user was assigned a unique user agent managing her preferences and recommendations.

3.1.2 Lessons Learned

The usage of agents to model the domain lead to a scalable and manageable system. New users are added by creating new user agents for them. The flexible service delivery schema made it easy to install redundant filtering agents for efficiency purposes being necessary after the number of users increased from 20 in the beginning to more than 200. However, with an increasing number of users we noticed a significant slow down in the overall answer behaviour of the system. This was mainly caused by a huge number of personal agents (one per user) that were running but inactive most of the time. The intuitive decision to provide one agent for each user proved to be a rather problematic design decision.

By suspending non-active agents automatically, the load on the system could be reduced to a minimum. The early overhead of modelling travel and profile data was compensated as soon as the system has been expanded with new features, travel agents, and information sources.

Another issue was the connection between user interfaces and the underlying system, as the user interface was event

[1]`http://www.eclipse.org`

oriented rather than goal oriented, and the synchronisation of the two models proved difficult.

3.2 PIA

PIA (Personal Information Agent) [4] is an ongoing project that started in 2003 with the aim to create a complete agent based solution for the personalised provision of information and news. Depending on an individual's personal interests and habits the systems provides relevant information at the right time. For example, it provides news in the morning and entertainment and event related information in the evening. Different filtering methods such as content-based and collaborative filtering [6] were implemented using agents. According to the particular needs, information is brokered to users using SMS, MMS, e-mail, or HTTP.

3.2.1 Setup and Methodology

The complexity was broken down into three tiers, one for the task of information extracting from different — typically external — sources, the other for information filtering, and the third for user personalisation and the provisioning of information. A highly distributed system with hundreds of agents — most of them being extractor agents — running on approximately 30 agent platforms was built. The most important part was the co-ordination filtering framework which resides on the filtering layer. Filtering agents utilising different methods of content-based and collaborative filtering techniques compete against each other in order to fulfill the users' information needs. So-called filter manager agents are monitoring, and controlling these filtering agents depending on information about the system state and estimations about the most promising filtering strategies. Additionally, rewards are computed on the basis of historical data, significance, as well as the users' feedback. As a result, the fitness of a filtering agent concerning a given information request changes over time. This yields to an adaptation of the system over time [5].

3.2.2 Lessons Learned

The PIA project is still ongoing, and has had a number of major revisions since its inception. The first project, while showing the power of using different agents to implement a variety of filtering and extraction functionalities, also showed the weaknesses of the system, some of which were conceptual, others related to the engineering approach. For example, extraction and filtering algorithms were computationally so expensive that they practically shut down agent communication on their nodes, leading to erratic behaviour as agents would not receive messages in time and therefore cancelled service calls. Different strategies of using the power of the agent paradigm, including using JIAC's planning capabilities to find good filtering strategies were working, but too slow for a production environment. The reaction time of the system grew with the number of users using the system, and was unacceptable even with a low number of concurrent users (a dozen). Another bottleneck was the directory facilitator (DF) implementation, as the amount of read and write calls lead to the system literally taking hours to start up. The problem of scalability and responsiveness could be traced back a combination of generic Java problems with prioritising threads on the one hand, and the design of JIAC IV to make heavy use of the DF as well as exchanging a lot of (often redundant) data within the meta protocol.

Another important problem was related to the operational availability and determinism. On the one hand the adaptation mechanisms made the system intentionally nondeterministic — leading to a high probability that an information request executed at two different time points would lead to different results. But often obscure results occurred as a consequence of unavailable content sources or crashed agents. In order to manage the large amount of agents, management functionalities were implemented and integrated into the agents, allowing for example automatic re-start in case of long timeouts. Also, administrators may manipulate these managed entities by use of a management console.

The current system, while still based on the idea of distributed filtering and extraction modules, is strongly limited in the number of agents. The dynamic delegation of filtering tasks to suitable agents in combination with agent mobility for load balancing purposes proved to be unsuitable for realtime behaviour. Therefore, once the co-ordination schemes were well understood, we concentrated related functionalities into one agent. Around 50 extractor agents perform the most crucial work of updating the content database which is now located on one server. All the filtering tasks are now controlled by one filtering agent which has the knowledge to decide which of the methods to apply for a given information request. I.e., all co-ordination that has been subject to inter-agent co-ordination mechanisms in the versions before, are now subject to internal control of an agent.

3.3 Common Criteria Certification

In order to establish whether the security features of JIAC are indeed industry grade, we did a security certification following the Common Criteria [1, 2, 3]. The Common Criteria are a worldwide accepted security standard in information technology. A defined process is used to check for security holes and issues. The security features are checked by independent reviewers and certified by an accredited institute. JIAC IV has been tested for security leaks by reviewers of T-Systems and certified by the*Bundesamt für Sicherheit in der Informationstechnik*, the federal office of IT security. The fact that the whole process took about two years and cost more than one million euros gives an idea of the scope of the certification process.

The security certification of JIAC IV was special in two respects. On the one hand, our research results had to be adapted to industry standards. On the other hand, it was a learning experience not only for us but also for the reviewers and certification institution because it was the first time that they had to certify a product with generic security features, rather than a concrete application.

3.3.1 Setup and Methodology

During a certification process, the reviewers examine the target product according to the criteria defined by the Common Criteria. They examine weaknesses within the functionality of the system, but also within the development environment and lifecycle of the product. The criteria are identical for all product classes and only differ in the evaluation level (also called EAL). The EAL defines the level of detail with which the criteria have been examined (For example, checking source code versus only checking use-cases). The choice of evaluation level reflects how detailed the evaluation has been, and has to be mentioned together with the certification. In the case of JIAC IV we chose EAL 3. This

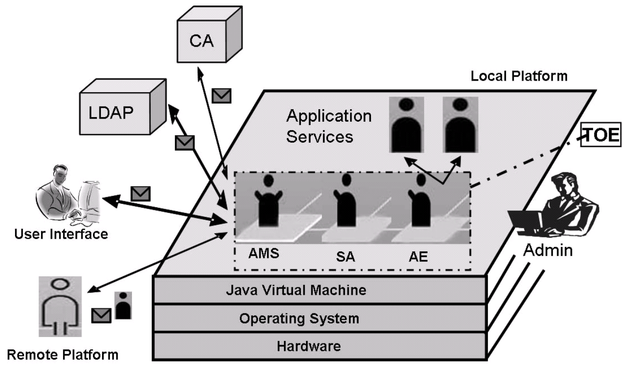

Figure 3: The target of evaluation

means that the reviewers executed tests independently from us, and checked the platform as well as the development environment for weaknesses on a very detailed level.

Because the certification process examines all parts of the product for weaknesses, we used a scaled down version of the agent framework, as shown in Figure 3. If we had not reduced the functionality of the framework, the effort needed to execute the certification would have been too much, not only for us but also for the reviewers and certification authority. This however was the result of the two-year long process of the certification, as we originally had planned to certify the whole framework. While the reduction was made in order to make the associated costs manageable, the effort involved in scaling down the framework was quite extensive too. For example, some documents describing features like high-level design, security targets, and more, had 300 iterations before the reviewers where content. The large amount of iterations between us and the reviewers was the result not only of the adaptation of the target platform, but also the problem of adapting the fairly high-level descriptions of the Common Criteria to our framework. Generally, only static security targets are certified, so that the dynamic aspect of JIAC IV was something that forced the reviewers to adapt the criteria.

3.3.2 Lessons Learned

There are a number of remarks that need to be made here. On the one hand, it appeared that the requirements of security targets in the "real world" are not easily adapted to a research environment. On the other hand, many processes generally used had to be adapted to cater for the concept of generic security, as they did not cover the special cases. For example, the requirement of having tests cover 100% of the code is simply not possible with generic security [17]. Generally however we can state that the security functionality of our framework has been certified according to internationally accepted standards. Also some of the requirements that are given in an industry setting but that do not necessarily make sense in a research environment have been adapted by our institute. For example, we learned that extensive tests do not necessarily mean an increase in time for the implementation, but can actually shorten the development time and make the product more stable.

In summary, we learned the following from the certification process:

- The JIAC IV agent framework can cope with international industry standards and requirements.
- The transfer of research results into industry was quite hard.
- Commercial security requirements can be met by agent frameworks.
- Common processes and best practices of quality assurance make sense and support a product design that has an industrial focus.

3.4 Nessi

A currently running project of our lab is the Network Security Simulator (Nessi) [11]. In the context of this project, devices within a large telecommunications network and possible threats are simulated. Using the simulator, the behaviour of attackers, threats, and possible counter measures can be evaluated.

3.4.1 Setup and Methodology

While the initial model used one agent per device for the simulation, the current implementation maps subnets onto agents. This change in the design was necessary to allow the system to scale to large networks of thousands of devices. Currently we simulate systems with about 100 subnets and about 3500 single devices.

One chief advantage of using agents in this project is the ability to simulate life cycles of the devices very easily. Furthermore, the abilities of agents support the requirements of the modelling of the simulated devices.

3.4.2 Lessons Learned

Depending on the target application, the "obvious" mapping of devices to agents is not necessarily the smartest one. Using aggregations allows to scale the system without loosing the advantage of agents or having less usable results.

4. CONCLUSION

There were a number of other projects implemented using JIAC IV, the ones above give a reasonable overview over the type of projects, as well as the experience we made. In this section we will try and summarise the pros and cons of using agents in general, and JIAC IV in particular.

One recurring theme that maybe did not come out clearly in the project descriptions was the disconnect between designing the system, and providing convincing and easy to use user interfaces (UI). There are two issues here that need to be addressed. The first is that for industrial projects, the presentation layer is often as important as the underlying framework — it is therefore not sufficient to provide functional but unpolished interfaces. This is nothing particular to agent based systems (other than sometimes the lack of responsiveness if a web-query requires different services to be executed. While the communication overhead does contribute to the high latency, it often comes with the territory of complex applications.) The second issue is more relevant to agents, and pertains to the actual human-computer interaction. Within a system here different agents interact and require user input in different situations, where users should take advantage of the flexibility of the system, providing a UI that allows for this flexibility while being easy and intuitive to use is a very difficult problem. Different paradigms, namely the task-oriented versus the goal oriented paradigm clash, and there are no easy ways to combine the two.

Another common complaint was the need to use JADL to program agents. Developers that were not familiar with the agent concept tended to write trivial services and plans, and instead moved the system logic into agent beans, thereby effectively circumventing advantages that JIAC IV provided in lieu of being able to program in their familiar language Java. We have tried to make the use of JIAC as simple and intuitive as possible by providing powerful tools, but the tools do not take away the fact that developers need to understand the agent- and service paradigm and be familiar enough with logical expressions to design pre-conditions and effects.

When implementing projects with dozens or hundreds of agents running on a number of platforms, another issue showed. While the distribution of agents allowed for scalable and flexible systems, the problem of deployment — getting everything running in the first place — was and is an often neglected problem. In most of our projects we put a lot of effort into integration and deployment management to ensure that any system built would be executable on the target system. Typical issues include the provision of all relevant dependent libraries (often different developers would use slightly different and slightly incompatible versions of the same library), adapting the system to the target network (including the configuration of firewalls and subnets), and the compatibility across different operating systems. Another side effect of large systems was that it became a challenge to keep track of running and dead agents over different platforms. Eventually we implemented a platform monitor that allowed the monitoring of agents.

Another issue in the context of scalability is the application of methodologies to the problem at hand. While often it is straightforward and elegant to map entities directly to agents, we found that when scaling application to serve hundreds or thousands of users, or incorporate a large number of functionalities, scalability broke down (in terms of number of agents as well as communication overhead). In a number of projects we therefore ended up mapping aggregations of entities to agents in order to keep the system load manageable.

One very prominent feature of JIAC IV, the ability to exchange components of agents during run-time, was never used. Also the planning component, that allowed agents to build plans from first principle never made it into a running project, mainly because the requirements were known, and there was never enough time to implement alternative service solutions that would have been needed to actually find alternative plans. We did however employ the planning in small example projects.

It must be said however that in spite of all the mentioned issues and problems, we found that JIAC IV has met industry standards and has successfully been applied in many different domains, ranging from healthcare and entertainment to information retrieval and personalisation to simulation and high security applications.

4.1 JIAC TNG — the Next Step

Based on our experience of using JIAC IV in large industry-driven projects, we are currently working on the next major release. There, a number of changes are introduced. First of all, we defined a number of agent types of different complexities to cater for the wish of being able to use simple lightweight agents together with intelligent goal-oriented agents in one system. One of the reasons for performance issues in our projects was the fact that even for simple reactive tasks, a developer had to employ a goal oriented agent, which of course required more resources than necessary. Additionally, we will allow a simple message based communication as an alternative to the service metaphor, because even though the meta-protocol proved quite useful, it was too much overhead for certain situations (e.g. simple inform-type messages).

Furthermore, we want to rework some of the core implementations of JIAC in order to achieve better stability and performance. While we think that the concepts and ideas of JIAC IV are mostly still good and appropriate, much has happened on the technical level. There are for example new libraries for component systems or threading, and we think that these will greatly improve the stability of our agent platform.

Our experience in recent projects is that a lot of emphasis is put on the aspect of composability of functionality on a very abstract level. Following the service oriented architecture approach [14] and business process modelling (for an overview see e.g. [21]), we are working on a mapping from BPMN (Business Process Modelling Notation) [23] to our agent framework [13, 12] in order to allow for high-level abstract design of an agent system that later can automatically be transformed into a (more or less) running system. Three current projects explicitly focus on the "easy service creation" and the support of powerful tools to connect similarly powerful services to a workflow. Goal is to allow for a modelling of systems on a level that is abstract enough to be easily understandable yet flexible enough to be scalable and adaptive.

While we do emphasise the workflow aspect of applications, we do see the need to goal orientation, and are currently working on a language that allows the seamless integration of services, semantics, and goal oriented programming.

5. REFERENCES

[1] Common Criteria, part 1: Introduction and general model, version 2.1, Aug 1999.

[2] Common Criteria, part 2: Security functional requirements, version 2.1, Aug 1999.

[3] Common Criteria, part 3: Security assurance requirements, version 2.1, Aug 1999.

[4] S. Albayrak. The role of AI in shaping smart services and smart systems. In *KI 2007: Advances in Artificial Intelligence*, volume 4667 of *LNAI*, page 1. Springer, 2007.

[5] S. Albayrak and D. Milosevic. Situation-aware coordination in multi agent filtering framework. In C. Aykanat, T. Dayar, and I. Korpeoglu, editors, *The 19th International Symposium on Computer and Information Sciences (ISCIS 04), Antalya, Turkey*, volume 3280 of *LNCS*, pages 480–492. Springer, 2004.

[6] S. Albayrak and D. Milosevic. Strategy coordination approach for safe learning about novel filtering strategies in multi agent framework. In D. Yeung et al., editors, *Advances in Machine Learning and Cybernetics*, volume 3930 of *LNAI*, pages 30–42. Springer Berlin, 2006.

[7] S. Albayrak and D. Wieczorek. JIAC - an open and scalable agent architecture for telecommunication applications. In S. Albayrak, editor, *Intelligent Agents in Telecommunications Applications - Basics, Tools, Languages and Applications*. IOS Press, Amsterdam, 1998.

[8] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Havard University Press, Cambridge, MA, 1987.

[9] K. Bsufka. *Public Key Infrastrukturen in Agentenarchitekturen zur Realisierung dienstbasierter Anwendungen*. Phd thesis, Technische Universität Berlin, 2006. `http://nbn-resolving.de/urn:nbn:de:kobv:83-opus-13536`.

[10] Bundesamt für Sicherheit in der Informationstechnik. BSI-DSZ-CC-0248-2005 for Java Intelligent Agent Componentware IV Version 4.3.11 from DAI-Labor Technische Universität Berlin. Certification Report BSI-DSZ-CC-0248-2005, Bundesamt für Sicherheit in der Informationstechnik, Jan 2005.

[11] R. Bye, S. Schmidt, K. Luther, and S. Albayrak. Application-level simulation for network security. In *First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SimoTools)*, 2008.

[12] H. Endert, B. Hirsch, T. Küster, and S. Albayrak. Towards a mapping from BPMN to agents. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, and K. P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin / Heidelberg, 2007.

[13] H. Endert, T. Küster, B. Hirsch, and S. Albayrak. Mapping BPMN to agents: An analysis. In M. Baldoni, C. Baroglio, and V. Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies*, pages 43–58, 2007.

[14] T. Erl. *Service-Oriented Architecture:Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall, Indiana, USA, August 2005.

[15] Foundation for Intelligent Physical Agents. FIPA Agent Communication Language Specifications, 2002.

[16] Foundation for Intelligent Physical Agents. Interaction Protocol Specifications, 2002.

[17] T. Geissler and O. Kroll-Peters. Applying security standards to multi agent systems. In *Proceedings of the First International Workshop on Safety and Security in Multiagent Systems, Sasamas'04, AAMAS'04*, 2004.

[18] J. Keiser, B. Hirsch, and S. Albayrak. Agents do it for money — accounting features in agents. In M. Dastani, A. E. F. Segrouchni, A. Ricci, and M. Winikoff, editors, *ProMAS 2007 Post-Proceedings*, volume 4908 of *LNAI*, pages 44–58. Springer Berlin/Heidelberg, 2008.

[19] S. C. Kleene. *Introduction to Metamathematics*. Wolters-Noordhoff Publishing and North-Holland Publishing Company, 1971. Written in 1953.

[20] T. Konnerth, B. Hirsch, and S. Albayrak. JADL — an agent description language for smart agents. In M. Baldoni and U. Endriss, editors, *Declarative Agent Languages and Technologies IV*, volume 4327 of *LNCS*, pages 141–155. Springer Berlin / Heidelberg, 2006.

[21] F. Lautenbacher and B. Bauer. A survey on workflow annotation & composition approaches. In M. Hepp, K. Hinkelmann, D. Karagiannis, R. Klein, and N. Stojanovic, editors, *Semantic Business Process and Product Lifecycle Management. Proceedings of the Workshop SBPM*, volume 251 of *CEUR-WS*, 2007.

[22] D. Martin, R. Hodgson, I. Horrocks, and P. Yendluri. Owl 1.1 web ontology language, 2006. `http://www.w3.org/Submission/2006/10/`.

[23] Object Management Group. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01, OMG, 2006. `http://www.bpmn.org/Documents/OMGFinalAdoptedBPMN1-0Spec06-02-01.pdf`.

[24] J. S. Penberthy and D. Weld. UCPOP: A sound, complete, partial-order planner for ADL. In *Proceedings of Knowledge Review 92*, pages 103–114, Cambridge, MA, October 1992.

[25] R. Sesseler and S. Albayrak. JIAC IV - an open, scalable agent architecture for telecommunications applications. In *Proceedings of the First International NAISO Congress on Autonomous Intelligent Systems (ICAIS 2002)*. ICSC Interdisciplinary Research, 2002.

[26] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1114, Dec 1980.

[27] E.-O. Tuguldur, A. Heßler, B. Hirsch, and S. Albayrak. Toolipse: An IDE for development of JIAC applications. In *Proceedings of PROMAS08: Programming Multi-Agent Systems*, 2008.

[28] J. Wohltorf, R. Cissée, and A. Rieger. Berlintainment: An agent-based context-aware entertainment planning system. *IEEE Communications Magazine*, 43(6):102–109, June 2005.