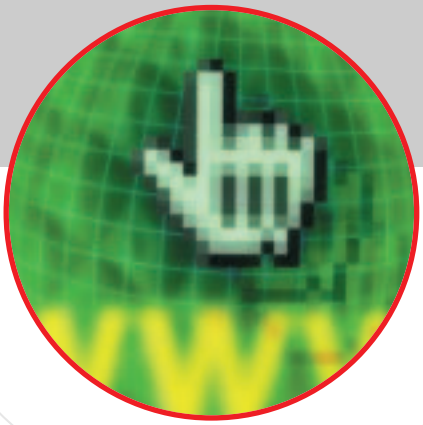


Jadex: Implementing a BDI-Infrastructure for JADE Agents

L. Braubach, W. Lamersdorf, A. Pokahr

ABSTRACT - Intelligent agents are a modelling paradigm, based on the notion of agents with mental states. The agent metaphor is nowadays used in many research and industry projects, and several generic agent platforms are available. Nevertheless, there is a gap between platforms concentrating on agent communication infrastructure and platforms concerned with the representation of internal agent concepts. This article presents an approach to bridge this gap: Jadex, an add-on to the widely used JADE agent platform. The add-on follows the BDI architecture, a well-known model for representing mentalistic concepts in the system design and implementation. The article provides an overview of the BDI model, and the design and realization in Jadex, as well as the integration of the add-on into the JADE agent framework.

BDI:
Belief Desire Intention

Introduction

Intelligent agents are considered a promising approach for building complex software systems, because the agent paradigm allows for modelling applications in a natural way that resembles how humans perceive the problem domain (1). Nevertheless, agent applications are still scarce in the market for several reasons. Firstly, the design and implementation requires skills in many different fields like distributed systems engineering, communication infrastructures and agent architectures. Secondly, there exists no general consensus for a modelling and methodology paradigm for multi-agent systems. In contrast a lot of different description techniques and methodologies were developed, but none of them has reached a high level of maturity yet (2). Thirdly, the choice of an agent-platform is crucial for the success of a software project, but difficult due to the large number of available platforms.

For developing agent systems it is necessary to consider the intra-agent as well as inter-agent structures. In this article it is argued that the BDI model is a sound foundation for modelling and implementing the internal agent behaviour. The BDI model enables to view an agent as a goal-directed entity that acts in a rational manner. To allow a smooth transition between the modelling and implementation phase the BDI paradigm has to be supported on the implementation level as well. Regarding inter-agent communication, the FIPA specification[1] can nowadays be seen as de-facto standard. Be-

1 http://www.fipa.org

pokahr@informatik.uni-hamburg.de

Alexander Pokahr

Alexander Pokahr received his diploma in computer science in January 2002 at the University of Hamburg. He worked in the field of model-based user interface construction systems and since April 2002 he is a research assistant at the University of Hamburg and deals with the modelling and implementation aspects of agent-based systems. The topic of his current project MedPAge is the agent-based management of hospital logistics. This work is funded by the Deutsche Forschungsgemeinschaft (DFG SPP 1083). Besides this work he gives practical courses in agent-oriented software engineering based on the JADE and Jadex technologies.

cause the JADE multi-agent platform (3) is an excellent choice due to its FIPA compliance and maturity, a generic approach to integrate the BDI concepts into JADE is presented.
In section "BDI Fundamentals" the BDI fundamentals regarding the individual concepts and their interplay are described. Section "Example" explains the design and implementation of the JADE add-on Jadex by showing the utilized mentalistic concepts and the integration into the JADE platform. Related BDI implementations are discussed in section "System Design and Realization". The article concludes with a summary and a description of the ongoing and future work.

BDI FUNDAMENTALS

The Belief-Desire-Intention model (BDI) was conceived by Bratman as a theory of human practical reasoning (4). Several software systems have been implemented, which are based on the BDI model, the most well-known representative being the Procedural Reasoning System – PRS (5). These systems reduce the abstract notions of desires and intentions to the more concrete concepts of goals and plans. The BDI model has later been formalized and further developed to the AgentSpeak(L) language (6). The three types of attitudes a BDI agent has, are now shortly sketched.
Beliefs are informational attitudes of an agent, i.e. beliefs represent the information, an agent has about the world it inhabits, and about its own internal state. But beliefs do not just represent entities in a kind of one-to-one mapping; they provide a domain-dependent abstraction of entities, by highlighting important properties, while omitting irrelevant details. This introduces a personal world view inside the agent: The way in which the agent perceives and thinks about the world.
The motivational attitudes of agents are captured in goals. The goal is a central concept of the BDI architecture, representing a certain target state that the agent is trying to reach. In a goal-oriented design, goals explicitly represent the states to be achieved, and therefore the reasons, why actions are executed. When actions fail it can be checked if the target state is already achieved, or if not, if it would be useful to retry the failed action, or try out another set of actions to achieve the target state. Moreover, the goal concept allows to model agents which are not purely reactive i.e., only act after the occurrence of some event. Agents that pursue their own goals exhibit pro-active behaviour.
Plans, which are deliberative attitudes, are the means by which agents achieve their goals. A plan is not just a sequence of basic actions, but may also include sub-goals. Other plans are executed to achieve the sub-goals of a plan, thereby forming a hierarchy of plans. The agent keeps track of the actions and sub-goals carried out by a plan, to determine and handle plan failures.

EXAMPLE

As a running example a simple translation agent is presented. The agent has the skills to translate a word from a source to a target language, to add new word pairs to its internal dictionary and to speak out sentences. It offers its abilities via a simple message-oriented service interface that complies to the FIPA request protocol.

In the following the example will be recalled, whenever it helps to understand certain details of the Jadex system or can be used to show how the presented concepts can be realized.

SYSTEM DESIGN AND REALIZATION

The JADE platform focuses on implementing the FIPA reference model, providing the required communication infrastructure and platform services such as agent management, and a set of development and debugging tools. It intentionally leaves open much of the issues of internal agent concepts. The JADE eXtension Jadex is an implementation of a hybrid (reactive and deliberative) agent architecture for representing mental states in JADE agents following the BDI model. It is designed for easy integration into JADE as an add-on package. The main objective is to facilitate the utilization of mentalistic concepts in the implementation, where this is regarded as appropriate by the agent developer.

Overview

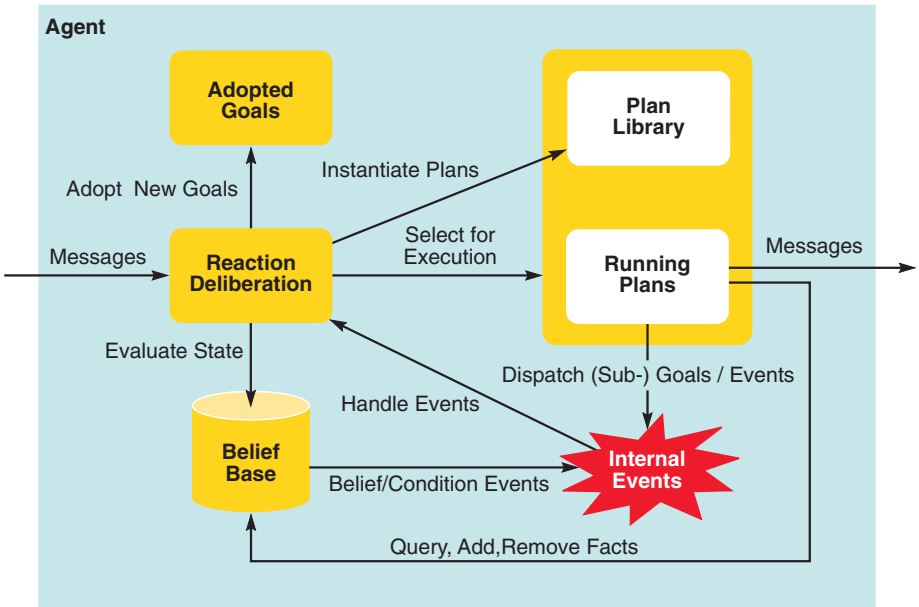
In figure 1 an overview of the abstract Jadex architecture is presented. Viewed from the outside, an agent is a black box, which receives and sends messages. Incoming messages, as well as internal events and new goals serve as input to the agent's internal reaction and deliberation mechanism. Based on the results of the deliberation process these events are dispatched to already running plans, or to new plans instantiated from the plan library. Running plans may access and modify the belief base, send messages to other agents, create new top-level or sub-goals, and cause internal events.

In the following the design and implementation of each of the concepts of the BDI architecture (beliefs, goals and plans) will be described. Afterwards, an overview of how a developer would apply these concepts to construct agents using the Jadex extension is given.

Beliefs

Each Jadex agent has a belief base to store the facts that make up the agent's knowledge. The

Figure 1
Jadex Abstract Architecture



facts are structured using slots representing the beliefs, which are named and typed, and can be either single-valued or multi-valued (containing ordered sets of facts).
Jadex does not require any special kind of knowledge representation, but allows arbitrary Java objects to be stored as facts. One option for the agent developer is to use a frame based knowledge representation, e.g., following the OKBC knowledge model (7). Using hand coded ontologies, or modelling tools like Protégé and code generation it is possible to capture the semantics of objects, to be stored as beliefs.
The belief base implementation also incorporates concepts from the relational database world. A set oriented declarative query language allows retrieving subsets of beliefs, or evaluating expressions over the belief base state. Another special feature of the belief base is the support for conditions. Conditions represent a persistent expression of a certain state e.g., of one or more beliefs. Once a condition is satisfied, an internal event is generated, which may trigger plans or plan steps, or lead to the adoption of new goals.
In the translation agent example a belief is used to represent the dictionary. This allows the translations plan as well as the plan which adds new word entries to access the dictionary. Moreover the translation plan makes use of the built-in query language for requesting words from the dictionary.

Goals

In Jadex three different kinds of goals can be distinguished: Achieve, maintain, and perform goal. The most basic kind of a goal, the achieve goal, just defines a desired target state, without specifying how to reach it. Agents may try several different alternatives, to achieve a goal of this kind. For goals of kind maintain, an agent keeps track of the state, and will continuously execute appropriate plans to re-establish the target state whenever needed. When not the desired target state, but rather the concrete actions to be done are the matter of subject, a goal is of kind perform. Perform goals directly specify the actions to execute, therefore an agent will not engage in any meta-level reasoning how to achieve a goal of this kind. Also, the success of the goal does not depend on the outcome of the actions, i.e., a perform goal succeeds even when a performed action fails (8).
Goals are represented as objects with several attributes. The target state of achieve goals can be explicitly specified by an expression (e.g., referring to beliefs), which is evaluated to check if the goal is achieved. Name and properties of the goal facilitate plan selection, and its parameters guide the actions of executing plans. A perform goal defines directly the plan that is executed. Both kinds of goals can be augmented with activation and deactivation condition, which are used to reproduce maintain goal like behaviour.
The execution semantics of an active goal is captured in so called BDI-flags, which guide plan selection and execution. The developer can choose if a goal should be given to only a single plan at a time, is retried after a plan failure, if failed plans should be excluded from the applicable plan list, and if meta-level reasoning is enabled.
Reasoning about goals can be implemented in two ways. Using activation and deactivation conditions, the agent developer can apply a rule based approach, to enable or disable goals when certain internal conditions (e.g. expressions defined over the belief base) hold. Alternatively, goals maybe activated and deactivated manually from procedurally implemented plans.
The translation agent registers its services using an achieve goal. Additionally it utilizes a conditional perform goal to clean up its word table, whenever hundred new entries have been added.

Plans

Jadex uses the plan-library approach to represent the plans of an agent, instead of performing ad-hoc planning. The agent programmer decomposes agent functionality into separate plans, which are implemented as Java classes. Therefore object-oriented techniques can be exploited in the development of plans. Plans can be reused in different agents, and can incorporate functionality implemented in other Java classes e.g., to access a legacy system.
The Jadex execution model is event-based. Everything happening inside a Jadex agent is repre-

sented as event. Message events denote the reception of an ACL message. Goal events announce the emergence and the achievement of goals, and internal events (called stimuli) report e.g., changes of beliefs, timeouts, or that conditions are satisfied.

Events trigger plan steps, by leading to an invocation of the action method of a plan. A plan step is executed as a whole, and may contain several basic actions and/or sub-goals. The action method of a Jadex plan is similar to the action method of a JADE behaviour, but is only called, when the event specific to the next plan step occurs. Plans create filters to wait for specific events, which trigger subsequent plan steps. In addition activation filters are used to specify which plan should be instantiated when a certain event occurs.

The event model allows already running plans to wait for specific goals to appear, which are then adopted by the plan. This is a fundamental difference to traditional PRS-style plans, which are created for a single root goal, and are deleted, once they have achieved, or failed to achieve the goal. In Jadex, so called service plans can be constructed, which cyclically adopt and process new goals. Service plans use permanent filters to announce interest in events they always want to handle, even when they are not currently ready to process them. Each running plan has its own internal wait queue, which collects events for the plan to process later. When no plan is currently interested in an event, the event is ignored and discarded. For the translation agent all of the plans except the speak plan are realized as PRS-style plans. This plan is a colourful example for a situation when it is advantageous to utilize a service plan, because its wait-queue can be used to keep the order of the incoming requests. Using a PRS-style plan would require a synchronization of multiple instances of the speak plan. Otherwise they would talk in disorder.

Agent Definition

To create and start an agent, the system needs to know the properties of the agent to be instantiated. The state of an agent is determined by the beliefs, the goals, the running plans, as well as the library of known plans.

Jadex uses a declarative and a procedural approach to define the components of an agent. The plan bodies have to be implemented as ordinary Java classes. All other concepts (beliefs, goals, filters, conditions) are specified using a language that allows for creating Jadex objects in a declarative way. If desired, inside the declarations the developer can refer to Java code e.g., defined in methods. The complete definition of an agent is captured in a so called agent definition file (ADF). In the ADF, the developer defines the initial beliefs and goals, by declaring the corresponding Java objects. Plans are declared by specifying how to instantiate them from their Java class. For plans to be instantiated on demand (called passive plans) the filter for the triggering event has to be stated. The filter is omitted in the case of a plan to be executed, when the agent starts (instant plan). In addition to the BDI components some other information is stored in the ADF e.g., default arguments for launching the agent or service descriptions for registering the agent at a directory facilitator.

The currently employed ADF format is a property file with name value pairs for mapping references to object declarations. To illustrate the agent definition in the property format, a cut out of the translation agent definition is depicted in figure 2 (keywords are **underlined**, "#" starts a comment). Because of the limitations of property files an XML format for specifying the ADF is under development.

Realization

To integrate the aforementioned BDI concepts into the JADE agent platform, several additional components are necessary. The core of a BDI architecture is obviously the mechanism for plan selection. Plans not only have to be selected for goals, but for internal events and incoming messages as well. To collect the incoming messages and forward them to the plan selection mechanism a specialized component is needed. Another mechanism is required to execute selected plans, and to keep track of plan steps to notice failures.

As a first step towards the BDI integration a rudimentary event handling and plan selection mechanism called dispatcher was developed. In a similar fashion to the approach described in (9) a

Figure 2
Translation Example Agent Definition

```
# start option
name                = TranslationAgent
class               = jadex.BDIAgent
# define plans
instant_plans       = espeak
plans               = egtrans, egadd, clean
# define plan mappings
espeak              = EnglishSpeechPlan()
egtrans             = EnglishGermanTranslationPlan()
egadd               = EnglishGermanAddWordPlan()
clean               = CleanWordTablePlan()
# define activation filters for passive plans
egtrans_filter      = EnglishGermanTranslationPlan.getEventFilter()
egadd_filter        = EnglishGermanAddWordPlan.getEventFilter()
# define initial beliefs and beliefsets
beliefsets          = egwords
egwords             = Tuple[]{Tuple("coffee", "Kaffee"), Tuple("milk", "Milch")}
# define goals
goals       = cleanup
# define goal mappings
cleanup     = Goal.createPerformGoal("cleanup", "addword_count%100==0", "clean")
```

JADE behaviour was implemented to collect messages and to invoke the dispatcher. For any occurring event, the dispatcher would immediately select a plan, which had to be implemented as JADE behaviour that is dynamically added to and removed from the agent.

This approach soon proved to be unsatisfactory, because among other problems the dispatcher could not keep track of the executing plans when using the standard JADE scheduling mechanism. In a redesign all of the required functionality was implemented in cleanly separated components. The relevant information about beliefs, goals, and plans is stored in data structures accessible to all of these components. The current implementation is described in the next section.

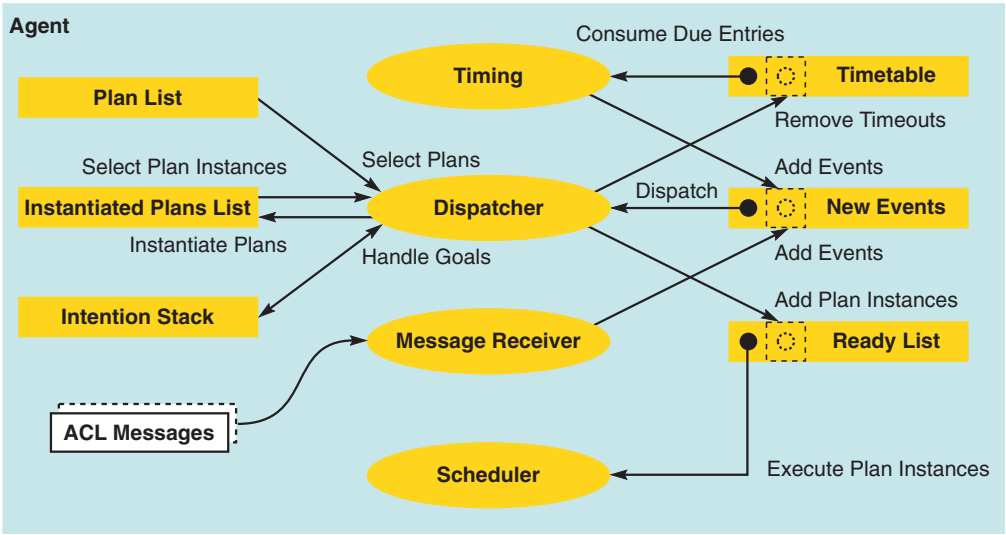
Execution Model

The functionality of the BDI architecture is implemented in four JADE behaviours, which run inside each Jadex agent: The scheduler, the dispatcher, the message receiver, and the timing behaviour (see figure 3). These behaviours operate concurrently on the internal data-structures of the agent. The message receiver and the timing process are very simple behaviours, with the single purpose to add new events to the event list. The message receiver listens for ACL messages from other agents, and creates corresponding message events. The timing behaviour removes events from the timetable, when their time point is reached, and appends them to the list of events to be dispatched. The dispatcher is responsible for adopting goals by placing them on the intention stack and selecting plans to handle events from the event list. The selected plans are subsequently executed step-by-step by the scheduler that also implements the plan supervision. The behaviours will put themselves to sleep, and restart each other appropriately, to avoid that the agent unnecessarily consumes CPU cycles, when it is actually idle.

Implementing the functionalities into separate behaviours provides a clean design and allows for flexible replacement of the behaviours with custom implementations, e.g. alternative scheduling mechanisms and BDI implementations can be tried out, using modified versions of the corresponding behaviours. The next section describes the default implementations of these two behaviours.

The dispatcher is the heart of the Jadex runtime system with the responsibility to select appropriate plans and plan instances to handle all events and goals inside the agent, facilitating the reactive and pro-active behaviour. It also manages the interre-

Figure 3
Jadex Execution Model



lation between plan instances and goals. The dispatcher cyclically removes the next entry from the event list, checks if a goal is associated with the event, and then creates the applicable plans list (APL) for the event. For events with goals, a meta-level reasoning will by default select the most appropriate plan from the APL. Internal events are forwarded to all applicable plans at once. When a goal is finished, i.e., either succeeded or failed, the owner of the goal will be notified. For a failed goal, the dispatcher may choose another plan for execution depending on the BDI flags of the goal.

The scheduler executes the ready-to-run plan instances one at a time, and step by step, applying an FCFS scheme. In each scheduling cycle, the first plan instance is removed from the ready list, and then a single step is executed. The scheduler waits until the plan step finishes or an error occurs. Afterwards it checks if any of the associated goals are already achieved. When it was the last step of the plan, the plan instance is removed from the agent.

JADE Integration

The jadex.BDIAgent class has been implemented as subclass of the jade.core.Agent. The BDIAgent initializes its beliefs goals and plans from an agent definition file. In addition it creates and starts the four internal behaviours, which provide the glue between JADE and the internal BDI execution process.

One aim of the Jadex project was to facilitate a smooth transition from developing conventional JADE agents to employing the mentalistic concepts of Jadex agents. All available JADE functionality can still be used in Jadex plans.[2] Moreover, it is possible to use some of the Jadex functionality e.g., the belief base or the goal stack, from conventional JADE behaviours. To use JADE behaviours in conjunction with Jadex plans the message receiver behaviour supports filtering of incoming ACL messages. It is necessary to sort out those messages which are handled by plans and therefore have to be dispatched to the internal Jadex system and keep the other messages available for the JADE behaviours.

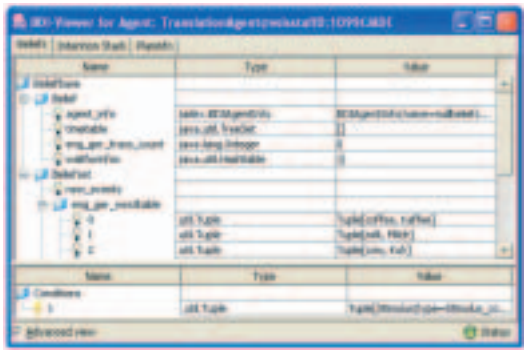
Development Tools

As a Jadex agent is still a JADE agent all available tools of JADE can also be used to develop Jadex agents. Most of the JADE platform deals with the ex-

2 Nevertheless, inside plans it is not recommended to bypass the dispatching mechanism by accessing the JADE message queue directly.

ternal view of an agent, which does not differ between conventional JADE agents and Jadex agents. Only the JADE introspector agent is of limited use, because it only shows the four Jadex standard behaviours and not the agent's plans. To enable debugging of Jadex agents a so called BDIViewer is currently developed and will support a visualization of the internal BDI concepts (see figure 4). Moreover, it will allow for debugging through stepwise execution of plans and manipulation of the agent's beliefs.
Once the ADF property format has been replaced by XML, development tools will be added to visually create and check agent definitions.

Figure 4
BDIViewer Screenshot



RELATED WORK

Nowadays a bunch of different multi-agent platforms is available. For a short overview of roundabout forty currently available systems the reader can refer to (10). They range from one-man open source systems to expensive commercial platforms with IDE integration. For a multi-agent platform of general applicability two features are essential. The first key feature is the support for mentalistic concepts to enable the smooth transition from the design to the implementation phase. The second key feature for a multi-agent platform is the provision of a sophisticated communication infrastructure that complies with communication standards and therefore enables platform interoperability.
Considering the support of mentalistic notions it can be stated that no general consensus exists about the appropriate set of concepts. This is reflected by a number of agent language families, most notably Agent-0 (11), AgentSpeak(L) (6), and 3-APL (12). Three reasons account for using the BDI model as introduced with the AgentSpeak(L) language family. First, the models underlying the other two languages assume a time-sliced execution of agents that is inefficient when agents are idle most of the time. The BDI model in contrast supports event-based reactive behaviour as well as pro-active (goal directed) behaviour. Secondly, the BDI model is a popular model with a solid theoretical and philosophical foundation (4). Finally, evidence exists that systems based on the BDI model can be used to build successful applications (13).
When considering the communication infrastructure of a platform it is essential that it complies with established standards, e.g. KQML or FIPA-ACL (14) Without a standard compliant communication and platform layer interoperability e.g. for world wide accessible agent service networks, like Agentcities[3], cannot be realized.
A comparison of the available platforms revealed that none of the systems fulfils both of these two key features equally well. Addressing the first key feature, Jadex was conceived not to "reinvent the wheel", but to combine the best concepts from well known BDI systems such as PRS (5), Jack (15) or JAM (8) and improve them with several new ideas. In contrast to these, Jadex is a pure Java API, therefore, developers do not have to learn a new language, and a wide range of existing development tools and IDEs for Java can still be used.
Moreover, Jadex is developed specifically to integrate well with the JADE platform, a FIPA-compliant agent platform, allowing Jadex to support the second key feature as well.
The only other approach bringing together FIPA-compliance and a BDI architecture is the FIPA-JACK system (16) that adds a FIPA compliant communication infrastructure to the sophisticated BDI platform JACK. Other approaches integrate a reasoning layer by making use of inference engines, e.g. JADE/Jess (17) or support

3 http://www.agentcities.org

planning based on goals, like PARADE (18). Comparing these approaches to Jadex several differences can be noted. In contrast to JADE/Jess and FIPA-JACK, Jadex exhibits an explicit representation of goals. In the opinion of many researchers goals have independent properties and should be distinguished from pure events (19). PARADE focuses on planning from first principles, while Jadex offers an API for goal-oriented programming. Moreover the Jadex add-on as well as the JADE platform are open source projects, making them open for modifications and custom extensions.

Conclusions and Outlook

This article argues for the use of mentalistic concepts as found in the BDI architecture, not only at the design level, but also in the implementation of agent-based systems. An overview of the BDI model was given, and the design and realization of a BDI implementation was described, and compared to related work.

The objective of the Jadex project is to integrate a BDI architecture into the JADE agent platform. The Jadex add-on allows agent developers to design and implement JADE agents, which exhibit goal-oriented (as opposed to task-oriented) behaviour.

The Jadex project is also seen as a means for researchers to further investigate which mentalistic concepts are appropriate in the design and implementation of agent systems. One important research area of future work is to make use of the BDI semantics already contained in the FIPA ACL communicative acts for simplifying the design and development multi-agent systems.

The Jadex system is continuously evolving while it is utilized. Currently the system is used as the basis of the research project MedPAge (20), which deals with agent-based management of hospital logistics. Additionally, the system is used in a teaching course "realisation of distributed agent systems" at the University of Hamburg.

A first public release of the Jadex system is available at the projects home page **http://sourceforge.net/projects/jadex.**

Acknowledgements

This work is supported in part by the German priority research programme SPP 1083: *Intelligent Agents in Real-World Business Applications.*

About the Authors

Alexander Pokahr and Lars Braubach received their diploma in computer science in January 2002 at the University of Hamburg. They worked in the field of model-based user interface construction systems and since April 2002 they are research assistants at the University of Hamburg and deal with the modelling and implementation aspects of agent-based systems. The topic of their current project MedPAge is the agent-based management of hospital logistics. This work is funded by the Deutsche Forschungsgemeinschaft (DFG SPP 1083). Besides this work they give practical courses in agent-oriented software engineering based on the JADE and Jadex technologies.

Glossary

ACL	*Agent Communication Language*
FIPA	*Foundations for Intelligent Physical Agents*
OKBC	*Open Knowledge Base Connectivity*
PRS	*Procedural Reasoning System*

References

[1] N. R. Jennings. An agent-based approach for building complex software systems, Communications of the ACM, vol. 44, no. 4, pp. 35-41, 2001.

[2] C. Iglesias, M. Garrijo, and J. Gonzalez, A Survey of Agent-Oriented Methodologies (in "Proceedings of the 5th International Workshop on Intelligent Agents (ATAL-98)"), J. Müller, M. Singh, and A. Rao, editors, Springer-Verlag, 1999.

[3] F. Bellifemine, G. Rimassa, and A. Poggi, JADE – A FIPA-compliant agent framework (in "4th International Conference on the Practical Applications of Agents

and Multi-Agent Systems (PAAM-99)"), The Practical Application Company Ltd., London, UK, 1999.

[4] M. Bratman, Intention, Plans, and Practical Reason, Harvard University Press, 1987.

[5] M. Georgeff and A. Lansky, Reactive Reasoning and Planning: An Experiment With a Mobile Robot (in "Proceedings of the 1987 National Conference on Artificial Intelligence (AAAI 87)"), pp. 677-682, Seattle, Washington, 1987.

[6] A. Rao, AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language (in "Agents Breaking Away"), W. van der Velde, and J. Perram, editors, Springer-Verlag, 1996.

[7] V. Chaudhri, A. Farquhar R. Fikes, P. Karp, and J. Rice, OKBC: A Programmatic Foundation for Knowledge Base Interoperability (in: "Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)"), AAAI Press/The MIT Press, pp. 600-607, 1998.

[8] M. Huber, JAM: A BDI-Theoretic Mobile Agent Architecture (in "Proceedings of the Third Annual Conference on Autonomous Agents"), pp. 236-243, O. Etzioni, J. P. Müller, J. Bradshaw, editors, ACM Press, 1999.

[9] M. Oja, B. Tamm, and K. Taveter, Agent-based Software Design (in "Proceedings of the Estonian Academy of Sciences"), vol. 50, pp. 5-21, 2001.

[10] E. Mangina, Review of Software Products for Multi-Agent Systems, AgentLink, software report 2002.

[11] Y. Shoham. Agent-oriented programming, Artificial Intelligence, vol. 60, pp. 51-92, Elsevier Amsterdam, 1993.

[12] K. Hindriks, F. De Boer, W. Van der Hoek, and J.-J. Meyer, Agent Programming in 3APL, Autonomous Agents and Multi-Agent Systems, vol. 2, no. 4, pp. 357-401, N. Jennings, K. Sycara, and M. Georgeff, editors, Kluwer Academic publishers, 1999.

[13] N. Howden, R. Ronnquist, A. Hodgson, and A. Lucas, JACK Intelligent Agents - Summary of an Agent Infrastructure (in "Proceedings of the 5th ACM International Conference on Autonomous Agents"), Canada, 2001.

[14] Y. Labrou, T. Finin, and Y. Peng, Agent Communication Languages: The Current Landscape, IEEE Intelligent Systems, vol. 14, no. 2, pp. 45-52, 1999.

[15] P. Busetta, R. Ronnquist, A. Hodgson and A. Lucas, JACK Intelligent Agents - Components for Intelligent Agents in Java, AgentLink News Letter, January 1999.

[16] K. Yoshimura. FIPA JACK: A Plugin for JACK Intelligent AgentsTM, Manual, RMIT University.

[17] E. Friedman-Hill, Jess in Action: Java Rule-Based Systems, Manning Publications Company, 2003.

[18] F. Bergenti and A. Poggi, A Development Toolkit to Realize Autonomous and Inter-operable Agents (in "Proceedings of Agents Fifth International Conference on Autonomous Agents), pp. 632-639, 2001.

[19] M. Winikoff, L. Padgham, and J. Harland, Simplifying the Development of Intelligent Agents (in "Proceedings of AI 2001: Advances in Artificial Intelligence"), pp.557-562, Springer-Verlag, 2001.

[20] T. Paulussen, N. Jennings, K. Decker, and A. Heinzl, Distributed Patient Scheduling in Hospitals (in "IJCAI 03 Proceedings"), 2003 (to appear).

CONTACTS

Lars Braubach
Distributed Systems and Information Systems
University of Hamburg
Tel.: +49 4042883 2091 - Fax: +49 4042883 2328
braubach@informatik.uni-hamburg.de

Prof. Dr. W. Lamersdorf
Distributed Systems and Information Systems
University of Hamburg
lamersd@informatik.uni-hamburg.de

Alexander Pokahr
Distributed Systems and Information Systems
University of Hamburg
Tel.: +49 4042883 2091 - Fax: +49 4042883 2328
pokahr@informatik.uni-hamburg.de