1.1 Jadex - Engineering Goal-Oriented Agents

In previous sections of the book agents have been considered as software artifacts that differ from objects mainly in their capability to autonomously execute tasks and to decide for themselves if and how something will be done. Following this view it is possible to design autonomous entities that communicate to solve application problems by cooperation or negotiation. An even more abstract view can be obtained if, besides autonomy, one explicitly supports the *proactiveness* of agents. Proactiveness basically means that agents can have their own goals and have behavioral freedom in which ways the goals are pursued, i.e., which means are used to fulfill the goals.

Most software systems and therefore also software agents, are goal-oriented in the way that they pursue the design goals laid down by the developer of the system. In this model software artifacts only have implicit goals and are not aware of them. In contrast, proactiveness is closely related to the explicit representation of goals and other mental attitudes. By using explicit representations, agents can be made aware of their goals and have the opportunity to reason about them, e.g., to find alternative ways to achieve goals, giving up unreachable goals or adopting new ones given the right opportunities.

Explicit representations of goals and other mental attitudes have been advocated by many well-known researchers such as [Dennett, 1987; McCarthy, 1979; Newell, 1990] for a long time as they drastically affect the human ability to understand and predict the behavior of systems. This provides a natural abstraction layer on which software agents can be built.

Several approaches exist that propose different kinds of mental attitudes and their relationships. One prominent approach is the BDI model that was originally conceived by [Bratman, 1987] as a philosophical theory of practical reasoning explaining human behavior with the attitudes: beliefs, desires and intentions. The basic assumption of the BDI model is that actions are derived in a two-step process called *practical reasoning*. In the first step – (goal) *deliberation* – it is decided which

set of desires should be pursued in the current situation represented in the agent's beliefs. The second step – *means-end reasoning* – is responsible for determining how such concrete desires produced as a result of the previous step can be achieved by employing the means available to the agent [Wooldridge, 2000].

[Rao & Georgeff, 1995] adopted the BDI model for software agents by introducing a formal theory and an abstract BDI interpreter that is the basis of nearly all historical and current BDI systems. The BDI systems designed in the spirit of Rao and Georgeff's interpreter are also called Procedural Reasoning Systems (PRS), termed after the first successfully implemented system. The interpreter operates on the agent's beliefs, goals and plans which represent slightly modified concepts of the original mentalistic notions. The most significant difference is that goals are considered as a consistent set of concrete desires that can be achieved altogether, thereby avoiding the need for a complex goal deliberation phase. The main task of the interpreter therefore is the realization of the means-end process by selecting and executing plans for a given goal or event.

This section describes one of these interpreters – the Jadex BDI reasoning engine that allows developing rational agents using mentalistic notions in the implementation layer. In other words it enables the construction of rational agents following the BDI model. In contrast to all other available BDI engines Jadex fully supports the two-step practical reasoning process (goal deliberation and means-end reasoning) instead of operationalizing only the means-end process. This means that Jadex allows for building agents with explicit representation of mental attitudes (beliefs, goals and plans) and that automatically deliberate about their goals and subsequently pursue them by applying appropriate plans. The reasoning engine is clearly separated from its underlying infrastructure, which provides basic platform services such as life cycle management and communication means. Hence, running Jadex on top of JADE combines the strength of a well-tested agent middleware with the abstract BDI execution model.

For the programming of agents, the engine relies on established techniques, such as Java and XML and, to further simplify development task Jadex includes a rich suite of runtime tools that are based

upon the JADE administration and debugging tools. It also includes a library of ready-to-use generic functionalities provided by several agent modules (capabilities).

1.1.1 Jadex Architecture

In Figure **Fehler! Kein Text mit angegebener Formatvorlage im Dokument.**.1 an overview of the abstract Jadex architecture is presented. Viewed from the outside, an agent is a black box capable of sending and receiving messages. The interpreter handles *practical reasoning* via two components responsible for *goal deliberation* and *means-end reasoning*. The goal-deliberation is mainly state-based and has the purpose to select the current non-conflicting set of goals.

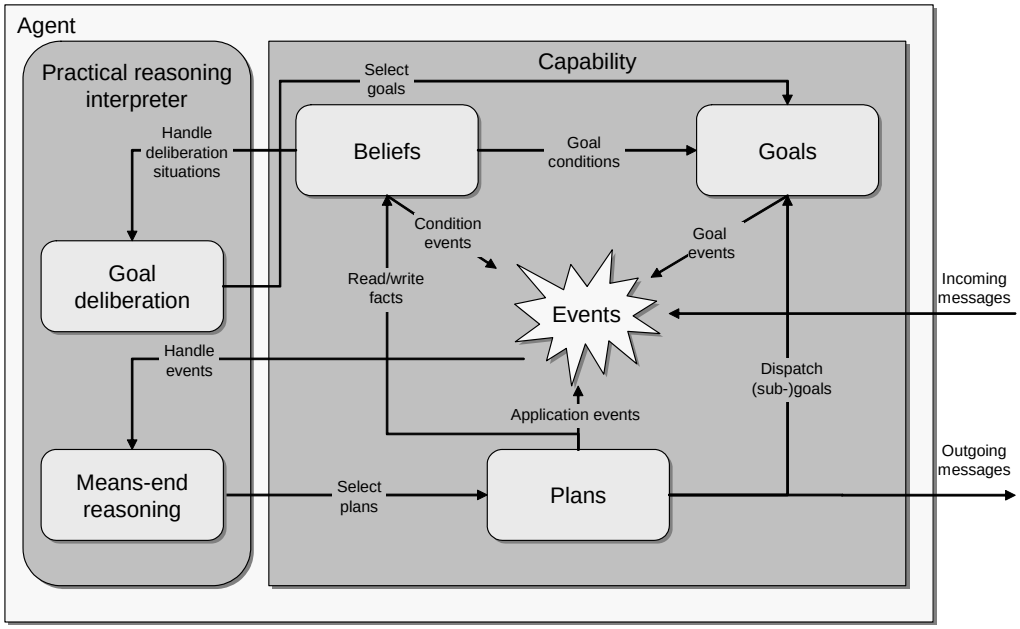

Figure Fehler! Kein Text mit angegebener Formatvorlage im Dokument**..1. Jadex abstract architecture**

Incoming messages, as well as internal events and new goals serve as input to the means-end reasoning component that dispatches these events to plans selected from the plan library for further processing. Running plans may access and modify the belief base, send messages to other agents, create new top-level or subgoals, and cause internal events. The interpreter represents the only global component within Jadex. All other components are contained in reusable modules called capabilities. In the following sections a short introduction to these programming concepts is

provided; for more detailed material the reader can refer to [Braubach *et al.,* 2005]. The internal mode of operation, which is quite different to traditional BDI systems, has been formally explained in [Pokahr *et al.,* 2005].

1.1.2 Agent Programming Concepts

Contrary to most other BDI systems Jadex intentionally does not promote a new agent programming language but instead relies completely on established software engineering techniques such as Java and XML. As a BDI agent consists of structural as well as behavioral parts, Jadex chooses a hybrid approach for defining and programming agents. The structural part comprises the agent's static design composed of elements such as belief, goal and plan types and additionally the agent's initial state consisting of e.g., goals and plans that will be pursued once the agent is activated. These structural aspects are specified in the Jadex XML language following an XML-Schema defining the BDI metamodel. On the other hand, the dynamic agent behavior needs to be encoded also. All such procedural knowledge is contained in the Jadex plans and is described using plain Java.

Figure **Fehler! Kein Text mit angegebener Formatvorlage im Dokument.**.2 illustrates the two conceptually different parts of a Jadex agent. On the left hand side is the agent definition file (ADF) for the agent type and on the right hand side are the procedural plans. The connection between these parts is established by an API enabling the Java plan classes to access BDI aspects such as reading the beliefs or issuing new subgoals.

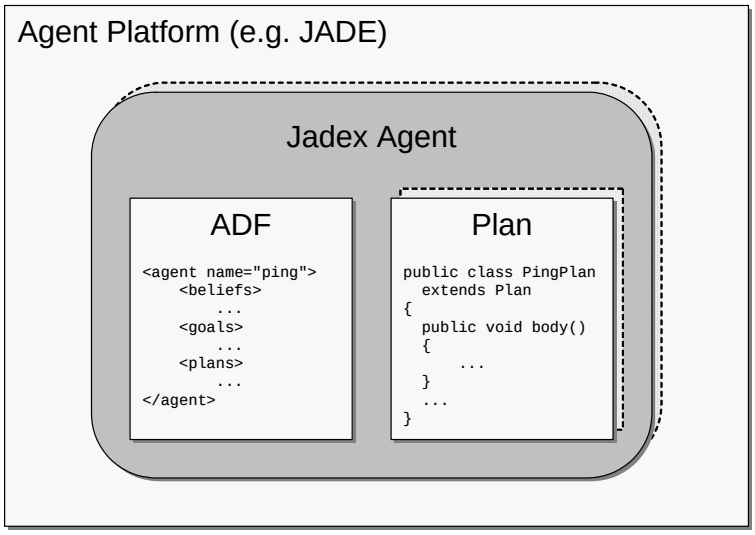

Figure Fehler! Kein Text mit angegebener Formatvorlage im Dokument.**.2 Jadex agent specification**

Beliefs

Beliefs represent the agent's knowledge about the world, itself and other agents. The belief representation in Jadex allows arbitrary Java objects to be stored instead of relying on a logic-based representation. This facilitates integration with existing software, e.g., classes generated by ontology modeling tools or database mapping layers can directly be reused. Objects are stored as named facts (called beliefs) or named sets of facts (called belief sets). Using the belief names, the 'beliefbase' can be manipulated by setting, adding, or removing facts. In addition, a more declarative way of accessing beliefs and beliefsets is provided by OQL-like queries. Besides being a passive data store for the agent the beliefbase also plays a vital role in the reasoning processes as changes in beliefs are automatically monitored by the engine and conditions can be defined referring, for example, to domain relevant belief values. Hence belief changes may trigger a goal's creation or drop condition, or render the context of a plan invalid leading to a plan abort.

Goals

Goals are the motivational force driving an agent's actions. They come in different flavors allowing various attitudes of an agent to be expressed. Jadex currently supports four application relevant goal types: perform, achieve, query and maintain goals. ***Perform goals*** express an agent's wish to directly engage in actions which are already satisfied if something else has been achieved. Such behavior is suitable for abstractly modeling activities such as driving around with a car just for fun. In contrast to such activity centric goals, ***achieve goals*** are associated with a desired world state. An example of this would be having the car parked close to its driving destination. In this case the state and position of a car is relevant for the goal's fulfillment state and hence expresses the goal's target world state. Therefore, achieve goals clearly emphasize the decoupling of what is to be achieved and how it will be brought about. ***Query goals*** instead can be used for information retrieval; their outcome is not defined by some target condition but by a query that needs to be answered. Depending on whether the agent's current knowledge plans may or may not be executed, meaning that if the information is readily available no additional work will be necessary. An example for a query goal is finding out the way to drive a car to an intended destination. Finally, ***maintain goals*** are used to describe situations that should be preserved by the agent under all circumstances. Strictly speaking, it means that whenever a specified situation is violated the agent will activate any applicable means to re-establish the desired world state. Ensuring that a car continues to function properly is an example for such a kind of goal. More information about the goal representation in Jadex can be found in [Braubach *et al.*, 2004].

When describing real world scenarios with goals the problem often arises that not all of the individual agent goals can be pursued at the same time. Such goal interferences are an important design aspect that is directly addressed by the goal deliberation facilities of Jadex. With the built-in *Easy Deliberation* strategy [Pokahr *et al.*, 2005] goal cardinalities and inhibition links between goals can be modeled at design time, with the system ensuring that during runtime only valid goal

subsets are active. If a goal set contains conflicting goals, the system will exploit the defined inhibition links to delay less important goals while executing the more important ones. Whenever goals are finished, the system considers the re-activation of currently inhibited goals.

Plans

Means-end reasoning is performed with the objective of determining suitable plans for pursuing goals or handling other kinds of events such as messages or belief changes. Instead of performing planning from first principles, PRS systems such as Jadex use the plan-library approach to represent the plans of an agent. A plan consists of two distinct parts: the plan head and the plan body. The plan head contains information about the situations in which the plan will be used. Most importantly it includes the events and goals the plan can handle and conditions that are used to restrict the applicability. Using conditions it is also possible to abort a running plan immediately if the current situation demands it. The plan body represents the recipe of actions that will be performed if the plan is chosen for execution. Depending on the plan's purpose its degree of abstractness varies continuously between very concrete and fully abstract. Concrete plans are fully specified at design time and consist of directly executable actions whereas only fully abstract plans are specified in terms of subgoals only.

Plan programming in Jadex requires the definition of the plan head in the ADF and the programming of the plan body in a pure Java class. The body is implemented by extending an existing Jadex framework class. This enables plan classes to access agent and BDI specific functionality such as sending messages, accessing beliefs or dispatching subgoals. Errors that may occur in BDI processing, such as a failed subgoal or a timeout when waiting for a reply message, are mapped to BDI exceptions causing the plan to fail if not intentionally caught by the programmer. Besides the main body, a plan may include special methods for clean up operations needed to properly complete a plan in the case of either failure or success.

Capabilities

A capability [Busetta *et al.*, 1999] results from the packaging of specific functionality into a module with precisely defined interfaces. An agent can be composed of an arbitrary number of capabilities that themselves may include any number of sub-capabilities. A capability description is defined in a separate XML document similar to the ADF and also consists of beliefs, plans and goals needed to generate intended behavior. Per default all capability elements have local scope and thus cannot be seen or used in other capabilities or in the agent. This ensures a maximum degree of encapsulation and avoids any kind of unintended interference between different capabilities. Those parts of the capability that constitute capability's interface need to be explicitly exported.

Jadex contains several generic plans and predefined capabilities in the package `jadex.planlib`. Basic platform features can be accessed by using the AMS and DF capabilities. The AMS capability offers goals for agent management such as creating new agents or destroying existing ones, whereas the DF capability can be used for accessing yellow pages services such as registering agents or searching specific services via goals. Furthermore, from the *protocols capability* several well-known FIPA interaction protocols such as request and contract-net are available via several goals. The rationale behind the goal oriented view provided also for protocols is that it allows adoption of a more abstract viewpoint that concentrates not on message flows but rather on the domain activities within a protocol.

Summary

In this chapter the Jadex BDI reasoning engine is presented. It allows for developing rational agents using mentalistic notions in the implementation layer. Outstanding features of the engine are the full support for the two phases of the practical reasoning process (goal deliberation and means-end reasoning), and the explicit representation of mental attitudes, such as beliefs, goals, and plans.

Moreover, the engine is specifically designed to build on and extend traditional software engineering principles and practices. Therefore, the engine is independent of the underlying infrastructure and can be flexibly deployed on middleware platforms such as JADE. For the

programming of agents, the engine relies on established techniques, such as Java and XML, allowing easy agent development in mature state-of-the-art development environments like eclipse and IntelliJ IDEA. To further simplify the development task, the Jadex distribution includes a rich suite of runtime tools for administration and debugging purposes as well as a library of ready-to-use generic functionalities provided by several agent modules (capabilities).

The Jadex BDI reasoning engine enables the construction of ***complex real-world applications*** by exploiting the ideas of intentional systems going back to Dennett and McCarthy. The high-level intentional view of the system-to-be can be preserved in the Jadex implementation leading to easy understandable and effectively manageable solutions.