Tools and Standards

Lars Braubach, Alexander Pokahr, Winfried Lamersdorf
University of Hamburg, Department of Computer Science, Distributed and Information Systems, {braubach | pokahr | lamersd}@informatik.uni-hamburg.de

Abstract. In this chapter tools, especially agent platforms, and relevant standards for realizing agent-oriented applications are presented. As there are a plenty of different agent platforms available the objective here is not to present an exhausting platform comparison but to introduce meaningful platform categories, relate them to existing standards and illustrate them with typical representatives. The categorization helps to understand the existing heterogeneous agent technology landscape and is one integral part of a proposed selection method. This method reflects the fact that different problem domains may demand very different solutions in terms of the used methodology and underlying agent platform. It sketches the important steps that can be used to find a suitable methodology and agent platform fitting to the problem domain at hand.

1 Introduction

This chapter discusses how the concepts of the previous chapters can be actually realized as part of a larger agent-based project. Given that most implementation details are to a large extent dependent on the concrete application requirements, this chapter can only provide general considerations regarding the selection of appropriate tools and standards. As the field of Agent Technology matures, tools and standards become an important success factor for the development of agent-based applications, as they allow drawing from the existing experience. Tools, most notably agent platforms, represent reusable implementations of generic technical requirements. Standards capture state-of-the-art knowledge and best practices.

2 From the problem domain to the implementation

The reason for selecting agent technology as part of a software project is mostly driven by the characteristics of the application domain at hand. [Weis2002] has identified some domain characteristics that advocate the

use of agent technology in general: Agents are a suitable technology and metaphor for the problem domain, when

- There is a dynamic number of components, i.e. the system needs to be open, allowing for new components to be introduced at any time.
- An external control of the entities comprising the system is not possible or not wanted, i.e. the system components have to be autonomous and self-dependent.
- The coordination within the system takes place by using complex communication relationships, i.e. for processes executed by the system complex interactions between the subcomponents of the system are required.

Among others, these characteristics are an important factor influencing the concrete decisions to be taken towards the transition from the requirements to an implemented system. Major decisions that have to be made regard the methodology to be followed (cf. Part IV, Chapter 1), and the agent platform to be used as a basis for the implemented system. The methodology guides the development process by proposing different development steps and the modeling artifacts being produced at each step. The agent platform forms the runtime environment for the agents that make up the application.

2.1 Criteria for selecting an agent platform and a methodology

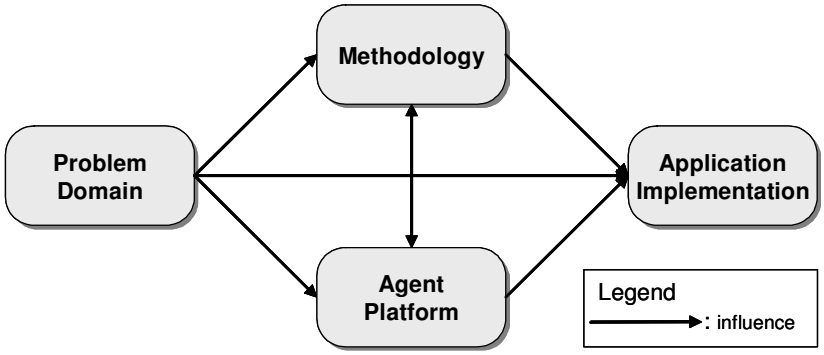

Figure 1. Influence relationships for application realizations

Decisions regarding both the methodology and the platform are influenced by the characteristics of the problem domain. Various catalogs of selection criteria have been proposed for comparing agent platforms (e.g. in

[BDDE2005] [EiMa2002]) and for comparing methodologies (see [SBPL2004]). The following presents some areas of domain dependant criteria considered most important as stated in [PoBL2005b]: *Concepts*, *Standards*, *Tools*, and *Applications*.

- Criteria in the area of concepts refer to the agent metaphor (e.g. deliberative entities vs. autonomous processes), and more specifically to details of the agent model (such as which mental attitudes are supported by a deliberative agent architecture).
- Relevant standards may come from two sources; on the one hand some standards are directly relevant to the problem domain (e.g. HL7[1] for health applications), on the other hand approved agent related standards (see Section 2.2) facilitate a consistent and interoperable design and implementation.
- Tool support has to address all phases of the development process starting from modeling the domain and elaborating the requirements to the system design and implementation. Implementation level tools can be further subdivided into code-oriented tools such as integrated development environments (IDEs), tools for debugging and testing, and tools for deployment and administration of an implemented system.
- Finally, examples of successful applications provide case studies of how to apply a certain approach and may reveal certain pitfalls.

Evaluation of these criteria is highly interrelated as these criteria apply to methodologies and platforms and to the problem domain as well. Therefore, the choice of an appropriate methodology and agent platform is crucial for the success of a project: The concrete platform determines the means, i.e. the concepts and supported standards that are available for system realization. Hence, it prescribes a certain agent philosophy, which has to be used for system implementation. If this agent philosophy does not reflect the important properties from the problem domain, a mismatch between problem domain and agent platform will complicate the realization. Such interdependencies also exist between the agent platform and the methodology. The methodology has to support the same agent philosophy, otherwise a mismatch between methodology and agent platform occurs, leading to a gap between modeling and implementation [SBPL2004]. Moreover, tool support not only for a methodology or a platform itself but also for mapping methodological design artifacts to a platform-specific implementation (e.g. code generators) further facilitates a smooth transition from design to implementation. Moreover, existing example applications of a methodology or platform allow to draw some conclusions perti-

[1] http://www.hl7.org/

nent to the given problem domain, e.g. regarding the context or size of the application.

As a result, most of the time a trade-off has to be made regarding concepts and standards. Some of them may match best to the problem domain but there may be insufficient support with respect to existing methodologies or agent platforms (see Figure 1). The availability and quality of tool support, as well as the existence of case studies describing successful (or failed) applications can further support the decision in favor of or against some methodology or agent platform. Finally, there is a number of other selection criteria which can be evaluated independently from the problem domain, such as the performance, availability (free or commercial), or usability of given tools, or the amount and quality of supplied documentation materials.

3 Agent platforms

An agent platform has the purpose to simplify the development of agent applications by providing an infrastructure agents can live in. It consists of the basic management facilities for hosting agents on a uniform infrastructure and additionally offers ready-to-use communication means for the agents. Agent platforms are characterized most notably by the internal and social architecture (layer 4 resp. 5 of the reference architecture from Part IV, Chapter 4) they employ. The internal architecture determines the internal concepts and mechanisms an agent uses to derive its actions whereas the social architecture is responsible for coordination between agents and team management. Technically, a platform is characterized by the programming language it provides for realizing agents and the available tools for development, administration and debugging.

In the remainder of this section an overview of existing agent platforms is given. This overview is not intended as an exhaustive list of all available platforms. For such a list the reader may refer to the "Agent Software" page of AgentLink[2] or (more focused on complete platforms) the agent platform page of the Jadex project[3]. Instead, this section will identify categories of platforms according to the reference architecture, highlight the important properties of these categories with respect to the above mentioned selection criteria, and present some representative platforms for each category. Finally, some general guidelines exemplify how to apply to

[2] http://www.agentlink.org/

[3] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/links.php

selection criteria to choose among the available platforms and methodologies.

3.1 Categorization of agent platforms with respect to the reference architecture

Referring to the reference architecture from Part IV, Chapter 4 a categorization of agent platforms can be done in accordance to the layers they emphasize (cf. Figure 2).

Considering an agent platform as a *middleware* for agent based services implies that at least L1-L3 need to be addressed in an adequate manner. *Middleware platforms* therefore provide a solid basis for developing open, interoperable agent systems, as they primarily tackle interoperability, agent management and communication means. Anyhow, not all important aspects of agent development are supported equally well. One important point that is not addressed to a satisfactory degree concerns the agent's reasoning process. Most middleware platforms rely on a simple task-based model that allows for programmatically composing complex behaviour out of simpler pieces.

Reasoning platforms focus on L4 and partly on L3 of the general reference architecture and hence employ an internal reasoning architecture for systematically deducing an agent's actions from some internal world knowledge. As the internal reasoning process often is intricate, support for L1-L3 greatly varies for different representatives. Additionally, middleware and reasoning platforms do not conceptually provide means for structuring and programming agent societies.

Social platforms address this issue by implementing organizational architectures (L5). An important question considering this kind of platform is, if the underlying architecture depends on the concepts provided by the internal architecture (L4). In this case, the cooperation and coordination mechanisms of the organizational architecture can be much elaborated allowing complex structures to be realized. On the other hand, the applicability of such an architecture and platform is restricted to agents conforming to a certain kind of agent type, such as BDI, which is undesirable for open system scenarios.

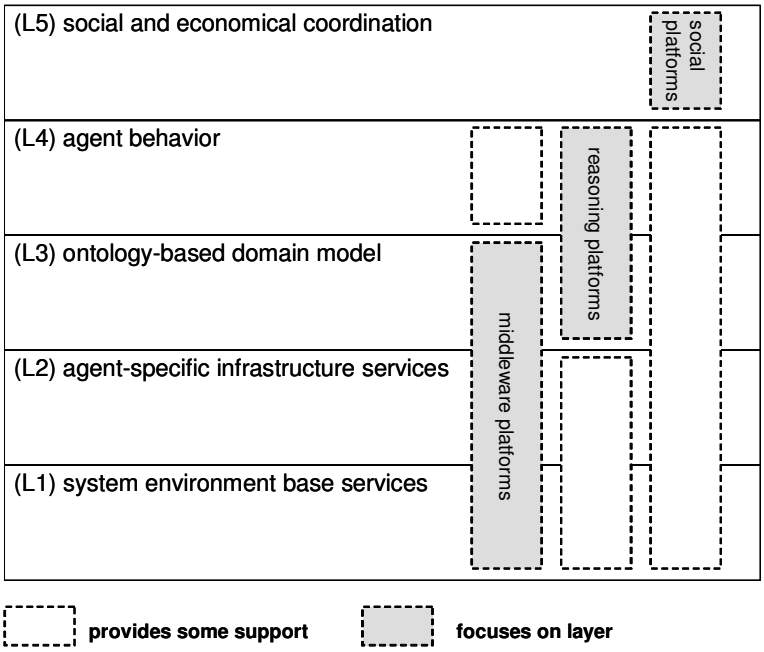

Figure 2. Coverage of layers for different categories of agent platforms

A common denominator for all the categories is the need for representing knowledge in an adequate manner (L3). At first sight this might be most interesting for reasoning platforms as they use the knowledge for internal deduction processes, but as communication plays a vital role in most multi-agent applications the need for exchanging knowledge is a predominant issue.

To capture the semantics of symbolic representations, ontologies can be defined. Ontology descriptions follow standards like RDF (http://www.w3.org/RDF/) and OWL (http://www.w3.org/2004/OWL/) (see Part IV, Chapter 1 and Chapter 2, Section 3). Ontology modeling tools such as Protégé (http://protege.stanford.edu/) allow creating and editing ontology specifications in the various standardized formats. Specialized reasoning engines such as RACER (http://www.racer-systems.com/) can be used to operate on the represented world knowledge, to derive new facts and possible courses of action.

3.1.1 Middleware platforms

In the field of distributed systems, *middleware* is normally seen as "[...] network-aware system software, layered between an application, the oper-

ating system, and the network transport layers, whose purpose is to facilitate some aspect of cooperative processing. Examples of middleware include directory services, message-passing mechanisms, distributed transaction processing (TP) monitors, object request brokers, remote procedure call (RPC) services, and database gateways."[4]

As agent orientation builds on concepts and technology of distributed systems, middleware is equally important for the realization of agent-based applications. Thereby, the term *agent middleware* is used to denote common services such as message passing or persistency management usable for agents. The paradigm shift towards autonomous software components in open, distributed environments requires on the one hand new standards to ensure interoperability between applications. On the other hand new middleware products implementing these standards are needed to facilitate fast development of robust and scalable applications. Agents can be seen as application layer software components using middleware to gain access to standardized services and infrastructure.

Before concrete examples of middleware platforms will be described the relevant middleware standards are introduced. Thanks to the FIPA standards the platform architecture has a common ground and interoperability between different middleware platforms could be achieved. Supplementary the MASIF standards define the basic concepts of agent mobility.

FIPA Standards

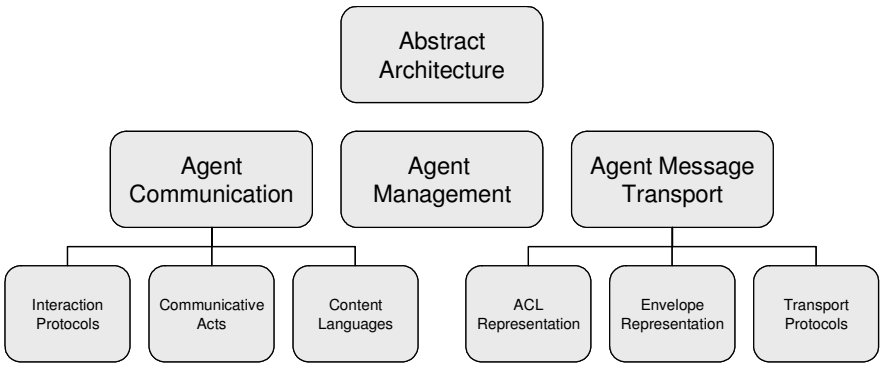

Figure 3. FIPA specification overview (from FIPA website)

An important foundation for the realization of middleware platforms are the specifications of the Foundation for Intelligent Physical Agents

[4] http://iishelp.web.cern.ch/IISHelp/iis/htm/core/iigloss.htm

(FIPA)[5] (see [PoCh2001]). The work on specifications considered application as well as middleware aspects. Specifications related to applications provide systematically studied example domains with service and ontology descriptions. The middleware-related specifications address in detail all building blocks required for an abstract agent platform architecture (see Figure 3).

The *abstract architecture specification* (FIPA00001) defines at a high level how two agents can find and communicate with each other. For this purpose a set of architectural elements and their relationships are described. Basically, two types of directories, for agents as well as for agent services, are introduced, which can be used by agents to register themselves or search for specific services. The communication between two agents relies on a message transport component, which has the task to send a message following the agent communication language (ACL) format. For agent communication and agent message transport many refining standards are available.

In the area of agent communication, various standards have been defined for diverse interaction protocols, communicative acts and content languages. Interaction protocols set up a context which constrains the possible course of interaction to predefined courses. Examples of interaction protocols include, besides others, Dutch (FIPA00032) and English (FIPA00031) auctions as well as the contract-net protocol (FIPA00029). The communicative act library specification (FIPA00037) describes the set of allowed performatives, which denote the meaning of a message according to speech act theory [Sear1969]. In addition different content languages can be employed for the representation of the message content. Examples include the FIPA semantic language (FIPA00008) and RDF (FIPA00011).

On the other hand the message transport has to deal with the representation of ACL messages and their envelopes as well as with the underlying transport protocols. For messages and envelopes different representations such as XML (FIPA00071/85) and a bit-efficient version (FIPA00069/88) have been proposed. Transport protocol specifications exist for IIOP (FIPA00075) and for HTTP (FIPA00084).

[5] http://www.fipa.org

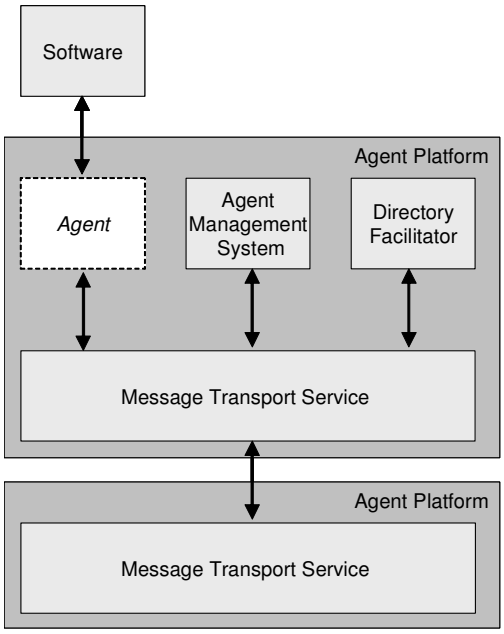

Figure 4. FIPA agent management reference model (from FIPA00023)

Most important for understanding the platform operation according to FIPA is the agent management specification (FIPA00023) (see Figure 4). It defines the necessary building blocks of an agent platform and their relationships, including mechanisms for agent management, as well as infrastructure elements such as directory services and message delivery. In this respect the agent management system (AMS) is responsible for exerting supervisory control over access to and the use of the agent platform. It maintains a directory of all agents living on the platforms. Another important component of an agent platform is the directory facilitator (DF) which provides yellow pages services to other agents. Agents hosted on a platform can access non-agent software and send messages to other agents on the same or another platform using the message transport service.

The FIPA specifications have been implemented in a number of agent platforms and interoperability among those platforms has been shown, for example in the agentcities network [WiCR2002].

MASIF Standards

The Mobile Agent System Interoperability Facility (MASIF) [OMG2000] is a standard for mobile agent systems proposed by the Object Management Group (OMG). The main objective of MASIF is to establish a common ground that allows MASIF compliant agent frameworks to perform

agent migration even in heterogeneous environments (assuming a common platform implementation language). It aims to achieve a first level of interoperability for the transportation of agent information where the information format is standardized. This means that once the agent data has been transferred the platform is responsible for interpreting the information. The transmitted data makes explicit the agent profile describing the language, serialization and further agent requirements on the platform. In this way MASIF enables an agent system to understand the agent's demands.

To achieve this kind of mobile agent interoperability MASIF tackles four different areas in the standard: agent management, agent transfer, agent / platform naming and agent system type / location syntax. Agent management concerns the life cycle control of agents including agents hosted on remote platforms. The management is addressed by standardized interfaces for agent creation and termination as well as for suspending and resuming agent execution. Agent transfer underpins the main goal of agent mobility and aims at a common infrastructure in which agents can freely move among different platforms. One necessary prerequisite for locating remote agents possibly hosted on another type of platform is that the syntax and semantics of agent and platform names are also standardized. In addition the agent system type is of importance as the agent transfer depends on the fact that the system can support the agent. Finally, the location syntax is standardized to ensure that platforms can find each other (cf. [Mil+1998] for details).

In addition to the functional aspects MASIF also tackles security issues arising in the context of mobile agents. An agent system has the task to protect its resources from new agents arriving at the platform. For this purpose the platform must be able to identify and verify the authority of an incoming agent. This allows for access control and agent authentication.

One big problem of MASIF is that it is based on CORBA and has therefore never been widely accepted. The MASIF standard has been used mainly for specialized mobile agent platforms such as Aglets [ClPE1997]. Nevertheless, also platforms supporting both FIPA and MASIF have been developed such as Grasshopper [BaMa1999].

JADE

A prominent example of a middleware-oriented agent platform is JADE (Java Agent DEvelopment Framework) [BCP2005], a Java framework for the development of distributed multi-agent applications. It represents an agent middleware providing a set of available and easy-to-use services and several graphical tools for administration and debugging. One main objec-

tive of the platform is to support interoperability by strictly adhering to the FIPA specifications concerning the platform architecture as well as the communication infrastructure. Recently, a "Web Services Integration Gateway" added support for agents acting as client or server in a Web Service application. Moreover, JADE is very flexible and can be adapted to be used also on devices with limited resources like PDAs and cell phones. The JADE platform is open source software, distributed by TILAB (Telecom Italia LABoratories). Since May 2003, an international JADE Board has the task of supervising the management of the project. Currently the JADE Board consists of five members: TILAB, Motorola, Whitestein Technologies AG, Profactor and France Telecom. Many JADE applications ranging from research prototypes to industrial products have been developed over the last years (see [BCP2005]). As one example Whitestein has used JADE to construct an agent-based system for decision making support in organ transplant centers [CFBB2004].

ADK

The agent development kit (ADK) is a commercial/open-source Java-based agent platform developed by Tryllian Solutions B.V. The main focus of the company is in the application integration area, involving all kinds of legacy system integration. In ADK, agent programming follows a task framework in which behavior is implemented as a set of simple tasks arranged in a workflow-like manner. The platform includes a visual design environment and administrative tools for deployment. The platform is targeted to be used in industrial systems (as opposed to research), and emphasizes mobility and security aspects. To facilitate the integration of legacy systems, interoperability with existing solutions is an important factor for the platform and a number of accepted industry standards are supported: SNMP (Simple Network Management Protocol) allows remote management of the agent platform. JNDI (Java Naming and Directory Interface) can be used for agent naming and lookup. Agents can receive messages sent using JMS (Java Messages Service), FIPA, or the JXTA peer to peer network. Moreover, agents can act as Web Service or interact with existing Web Services using the SOAP/WSDL/UDDI stack. Recently a business rule engine has been added, to support the maintenance of processes directly at the business level. Several production grade applications have been developed such as the "ePosit" system for intelligent Web search, or the "Continuous Auditing" system, which allows monitoring decentralized organizations and automating routine auditing tasks.

FIPA-OS

FIPA-OS was one of the first open-source FIPA-compliant software frameworks originating from research at Nortel Networks Harlow Laboratories in the UK. It is implemented in Java and like JADE uses a simple task-based approach as internal agent structure. Although development of FIPA-OS has been discontinued in 2003, the platform is still available for download. In addition FIPA-OS has been released as a reduced version suitable to small and mobile devices (MicroFIPA-OS). Tool support is limited to simple graphical user interfaces for administering and configuring the platform and agents on the platform. Up to now, FIPA-OS has been used mostly in research and beta stage prototype applications. For example emorphia Ltd. has developed an agent-based intelligent meeting scheduler named Friday based on FIPA-OS.

DIET

DIET Agents is a multi-agent platform developed as part of an EU project under the leadership of British Telecom. The DIET (Decentralized Information Ecosystem Technologies) project aimed at developing a lightweight, scalable, and robust agent platform targeted to peer-to-peer (p2p) and/or adaptive, distributed applications. Primary application area of the platform in the course of the project was information retrieval, filtering, mining and trading. The platform uses bottom-up, nature-inspired techniques from Evolutionary Computation and Artificial Life to provide an open, robust, adaptive and scalable environment for information processing and management. Tests performed by the project partners showed that the platform supports up to 100000 agents running on a single computer. After the project had finished in 2003 the platform was released as Open Source and is currently continued to be developed as a generic middleware agent platform. Besides the platform itself, a graphical tool for visualizing and debugging applications has been made available. Existing applications have mostly been developed in the course of the research project as prototypes and proof of concepts, e.g. for a collaborative tool visualizing social networks, self-organizing communities, and p2p content sharing applications.

3.1.2 Reasoning platforms

Reasoning platforms are based on specific internal agent architectures. Such internal agent architectures have been conceived to support the reasoning process of agents and therefore systematize the process of how an agent decides which action it wants to perform in any given situation. Ac-

cording to [WoJe2005] these architectures can be categorized into *reactive*, *deliberative* and *hybrid* architectures.

Reactive architectures abstain from any kind of symbolic knowledge and do not use symbolic reasoning. The most prominent reactive architecture is Brook's subsumption architecture [Broo1986] which assumes that an agent is composed of a hierarchy of task-accomplishing behaviors. Behaviors at a lower level in the hierarchy represent primitive actions and have precedence over higher-level behaviors. Even though the resulting agents are quite simplistic in nature it could be shown that this kind of architecture is well-suited for certain kinds of applications such as the movement control for robots.

Deliberative architectures require an agent having a symbolic model of the world and using logical (or at least pseudo-logical) reasoning for its decisions. Many deliberative architectures are based on a central planner component which is responsible for deducing reasonable agent actions. Examples of such architectures are IRMA [BrIP1988] and IPEM [AmSt1988]. Main drawback of most purely deliberative architectures is their inefficiency, as symbolic reasoning requires complex computations and thus cannot guarantee responsive agent behavior under all conditions.

To the rescue, many *hybrid architectures* have been proposed, which aim at bringing together the best from both approaches. Hybrid architectures combine reactive and deliberative facets leading to agent behavior that is responsive as well as intelligent. Even though there are no standards for reasoning facets of platforms two predominant architectures exist. Most influential architectures with respect to their practical relevance are the SOAR [LeLR1996] and the BDI [Brat1987] models of agency.

SOAR is based on Newell's psychological theory "Unified Theory of Cognition (UTC)" [Newe1990], which postulates the pursuit for a single set of mechanisms that account for all aspects of cognition such as memory, problem solving and learning. "A UTC must explain how intelligent organisms flexibly react to stimuli from the environment, how they exhibit goal-directed behavior and acquire goals rationally, how they represent knowledge (or which symbols they use), and learning."[6]

The BDI model was originally conceived by Bratman as a theory of human practical reasoning [Brat1987]. Its success is based on its simplicity reducing the explanation framework for complex human behaviour to the *motivational stance* [Denn1987]. Following the motivational stance, causes for actions are only related to desires ignoring other facets of cognition such as emotions. Another advantage of the BDI model is the consistent usage of folk psychological notions that closely correspond to the

[6] http://en.wikipedia.org/wiki/Unified_Theory_of_Cognition

way people communicate about human behaviour [Norl2004]. Starting from Bratman's work, Rao and Georgeff [RaGe1995] conceived a formal BDI theory, which defines beliefs, desires, and intentions as mental attitudes represented as possible world states. The intentions of an agent are subsets of the beliefs and desires, i.e., an agent acts towards some of the world states it desires to be true and believes to be possible. To be computationally tractable Rao and Georgeff also proposed several simplifications to the theory, the most important one being that only beliefs are represented explicitly. Desires are reduced to events that are handled by predefined plan templates, and intentions are represented implicitly by the runtime stack of executed plans. As a multitude of platforms have been developed based on the BDI paradigm, only a small selection is presented here. For a more detailed overview of BDI systems see [MaDA2005].

JACK

The JACK platform is developed as a commercial product by Agent Oriented Software [HRHL2001]. It is based on the BDI architecture and provides its own programming language called JACK agent language (JAL). JAL is a conservative extension of Java introducing BDI concepts and some features of logic languages such as cursors and logical variables. An agent in JACK is composed of a number of different JAL files, mainly representing the agent itself as well as its plans, beliefbase and events. To execute a JACK agent, its set of JAL files is first precompiled to Java source code and in a second step compiled to executable Java byte code. JACK addresses several weaknesses of traditional BDI systems. Most notably, it introduces the notion of a capability for the modularization of agents [BHRH2000]. Additionally, the SimpleTeams approach (see below) has been conceived to support the cooperation of agent within BDI teams. JACK represents an industry-grade product delivering extensive documentation and supporting tools. Especially, JACK ships with an IDE that supports the detailed design and implementation phase. The IDE supports inter alia the project management, the editing of files by syntax highlighting and the compilation and execution from within the IDE. Additionally a graphical plan editor allows for creating plans visually and observing their execution at runtime. It has been used in a variety of industrial applications as well as for many research projects. The application areas include Unmanned Aerial Vehicles (UAVs), human-like decision making and decision support systems (details can be found in [Wini2005]).

Jadex

Jadex [BrPL2005] [PoBL2005] is an open source software framework developed at the University of Hamburg. It allows the creation of goal oriented agents following the belief-desire-intention (BDI) model. The framework is realized as a rational agent layer that sits on top of a middleware agent infrastructure such as JADE [BBCP2005], and supports agent development with well established technologies such as Java and XML. Thereby, Jadex avoids intentionally the introduction of a new programming language and subdivides the agent description into structure and behavior specification. The structure of an agent is described in an XML file following a BDI metamodel defined in XML-schema whereas the behavior is implemented in plans that are ordinary Java files. This has the advantage that any state-of-the art IDE (offering XML and Java support) can be utilized for programming Jadex agents. Jadex introduces the basic concepts beliefs, plans, goals, events for agent programming and capabilities for modularization purposes. Besides the focus on middleware support, the Jadex reasoning engine addresses traditional limitations of BDI systems by introducing new concepts such as explicit goals and goal deliberation mechanisms (see e.g. [BPML2004], making results from goal oriented analysis and design methods (e.g. KAOS or Tropos) more easily transferable to the implementation layer. Besides the framework, additional tools are included to facilitate administration and debugging of agent applications. Jadex has been used to realize applications in different domains such as simulation, scheduling, and mobile computation. For example, Jadex was used to realize a multi-agent application for negotiation of treatment schedules in hospitals (see Part III, Chapter 4).

Jason

Jason [BoHV2005] is a platform for programming agents in AgentSpeak(L) [Rao1996], a logic-based agent-oriented programming language that is adequate for the realization of reactive planning systems according to the BDI architecture. In AgentSpeak(L) an agent consists of beliefs, represented as ground (first-order) atomic formulae, plans comprising basic actions and subgoal calls, as well as events that represent all kinds of relevant occurrences such as new goals or beliefs. Jason is a relatively slim BDI system strictly adhering to the formal AgentSpeak(L) semantics. This enables Jason to be used for model checking and verification purposes. The platform, which is available as open source, offers means for distributing an MAS over network and comes with a simple IDE for editing and starting agent applications. It has been used so far for several small academic applications.

SOAR

In contrast to the aforementioned reasoning platforms SOAR is not based on BDI, but relies on UTC [Newe1990]. The SOAR architecture at its heart is a typical production system that matches and applies rules on a working memory. It is enhanced with a learning mechanism called chunking [LeLR1996] which infers more abstract rules from observing the rule application process. On top of this production system a goal-driven problem solver following the problem space hypothesis is placed. SOAR utilizes an agent deliberation cycle consisting of the five phases: perceptual input, operator proposal, operator selection, operator application and output. In the perceptual input phase sensory data from the environment is updated and made available for the system. Next, in the proposal phase, productions fire to interpret the new data until no new data can be deduced (quiescence), propose operators for the current situation and to compare the proposed operators. In the selection phase, the operator to apply is chosen on basis of the proposed set of operators. When no unique operator is preferred, a so called impasse occurs and a new subgoal is created, which has the task to resolve the conflict (a process called *automatic subgoaling hypothesis*). In the application phase the selected operator is executed and finally in the output phase output commands are sent to the environment. The SOAR architecture for single agents is supplemented by a social architecture for agent teams (see below). The SOAR platform comes with an extensive tool support, documentation and example applications. VisualSoar is a simple form of an IDE specifically tailored to support writing SOAR agents and execute them in the runtime environment. In addition a SOAR debugger tool is provided for observing the internal data and behaviour of an agent. SOAR has been used in many projects, ranging from simple research to complex commercial application scenarios. As an example Soar Technology Inc.[7] uses SOAR agents for building various (e.g., pilot) training applications.

3.1.3 Social platforms

Social agent platforms provide support for expressing group behaviour within multi-agent systems. These systems build upon different group behaviour theories and architectures, which will be discussed next. Fundamentally, teamwork involves the structural as well as behavioural dimension. Nevertheless, current research does not provide integrated theories covering both dimensions at a satisfactory degree within one coherent

[7] http://www.soartech.com/

framework. Hence, in the following both aspects will be discussed separately.

One very simple, but nonetheless influential, structuring mechanism for agent teams is the Agent-Group-Role (AGR) model [Ferb2003]. Basically, an agent is seen as an active, communicating entity playing roles within groups. A group in turn is described as a set of agents sharing some common property. It is used as a context for a pattern of activities, and subdivides organizations. Agents are only allowed to communicate, if they belong to the same group. A role is the abstract representation of an agent's functional position in a group. An agent must play a role in a group, but an agent may play arbitrary many roles. One of the basic principles of the AGR model is that at the organizational level no agent description and therefore no mental issues should be used. This makes AGR independent of any particular agent model (in L4) and allows simple agents as well as very complex agents, possibly employing the intentional stance, being part of the same organizational structure. There are some approaches to standardize the structural aspects of teamwork; most notably the role concept and related terms specified as part of the AUML, which has many similarities to the AGR model (see [OdPF2003] for details).

The most influential framework for describing the behavioural aspects of teamwork is the joint intentions theory [CoLe1991]. It formulates the formal principles for describing how agents can pursue a common goal relying on the basic concepts beliefs, goals and their collective counterparts as foundations. The notion of a joint intention is regarded as a joint commitment of some agents to perform a collective action while being in a certain shared mental state. The joint commitment to perform some action is thereby represented as a joint persistent goal shared by all involved agents. One important property of such a joint goal, in contrast to an individual goal, is that the participating agents agree to inform each other about a possibly changing goal state. This means that each individual agent accepts responsibility for the pursuit of the common goal and informs the other if it e.g. finds out that the goal is unachievable allowing others to share that knowledge. Despite its neatness, the joint intentions theory does not address some important aspects. It is not discussed how agents can establish a joint intention towards some action. Also, the defection of a single agent causes the entire group task to fail. For these reasons the joint intentions theory was subject to several extensions which tried to expand and enhance the basic model. Examples for such extensions are Jennings' joint responsibility theory [JeMa1992] and Tambe's STEAM model [Tamb1997].

MadKit

MadKit is a modular and scalable multi-agent platform developed by Ferber and colleagues [GuFe2001]. It is built upon the AGR (Agent/Group/Role) organizational model, in which agents are members of groups and play certain roles. As the AGR model is independent from the underlying internal agent model, it allows a high heterogeneity of agent architectures and communication languages to be used. The MadKit platform is realized by following three design principles. Firstly, the system is based on a micro-kernel architecture that provides the basic services for agent resp. group management and message passing. Secondly, most services within MadKit are realized as agents making the system structure very flexible. Thirdly, MadKit provides a component oriented model for displaying agent GUIs within the platform. The tool support for the platform is quite extensive and comprises a graphical administration as well as several monitoring and debugging tools. The platform has been used for the realization of various applications such as TurtleKit, an agent simulation environment and SEdit, a tool for the design and animation of structured diagrams.

STEAM

STEAM [Tamb1997] is a general model of teamwork conceived to support performing coordinated tasks within some group of agents. It utilizes the formal joint intentions theory as basic building block, but borrows some ideas from the SharedPlans theory as well [GrKr1996]. Moreover, STEAM proposes several improvements regarding practical issues for making the model efficiently implementable. STEAM introduces team operators (team activities) and team beliefs as new concepts. Whenever a team activity needs to be executed, the agents belonging to the relevant team must first establish a joint intention for this team activity. To achieve a joint intention, an “establish commitments” protocol is carried out. After the joint intention has been established, a team operator can only be terminated by modifying the team state (mutual beliefs within this team). Conditions describing success and failure states can be specified for team operators individually. STEAM automatically takes responsibility for updating the team state whenever an important change occurs within a local view of a team belief. In this case the corresponding agent broadcasts this change to all other team members that update their view accordingly. In case of a failure during the team activity STEAM provides also means for replanning the task. For this purpose the contributions of the team members for a team operator are specified in terms of roles. A role is considered here as the set of activities an individual or subteam undertakes in service of the team’s overall task. STEAM allows specific role relationships being specified

(and, or, depends on) that are employed to determine the state of a team operator and possibly to engage into repair operations, e.g. substitute some critical subteam. STEAM has been implemented for the SOAR agent platform as a set of rules. This implementation has been used for diverse application domains, including RoboCup soccer and simulation environments for training.

JACK SimpleTeams

The JACK SimpleTeams approach [HoRB1999] aims at providing coordinated activities for groups of agents. It is based on the idea that a team is itself an autonomous entity that can exist without its team members and can reason about its behavior. The approach is an extension conceived specifically for BDI agents and adds new team constructs for roles, team plans and team beliefs to the standard BDI concepts. A team is represented as an extended BDI agent that is capable to cope also with these new concepts. The structure of a team is described with roles. More precisely, it is characterized by the roles it performs and the roles it requires others to perform. Thereby roles are used as abstract placeholders for arbitrary team instances playing that role at runtime. For this reason roles can be seen as a kind of interface for teams. Concretely, a role defines the relationship between teams and subteams in terms of the goal and belief exchanges implied by the relationship. The tasks of a team can be specified via team plans that extend the plan concept of BDI agents and enable coordinated task achievement. A team plan can be used to accomplish some task collaboratively by a (sub) set of agents belonging to the team. Therefore, a team plan offers possibilities to influence the actual selection of agents working on the task and new means for the distribution of subtasks to the participating members. The distribution of subtasks is done by subgoal delegation to team members. This allows the team members to decide in their own responsibility how to accomplish the goal retaining the full flexibility of multi-agent systems. To enable easy information exchange between the team members and the team itself, the concept of team beliefs is introduced. Team beliefs can either distribute data from the team to the subteams or aggregate data from the members back to the team. At runtime, a team runs through different phases. In the initial phase, the team formation is performed. This means that role fillers for all roles within the team are searched. When this formation ends the team enters the operational phase, in which the actual task processing is done. JACK SimpleTeams is a general purpose teamwork architecture. Nevertheless, it was used primarily for military application scenarios so far.

3.2 Platform summary

	Concepts	Standards	Tools	Applications	Availability
JADE	M	FIPA, WS	A, D	Production	Open source
FIPA-OS	M	FIPA	A	Beta	Open source
ADK	M	FIPA, WS, JMS, …	I, A	Production	Commercial
DIET	M	-	D	Beta	Open source
Jadex	M, R	FIPA, WS, JMS	A, D	Production	Open source
Jason	R	-	I, D	Beta	Open source
JACK +Simple-Teams	R R, S	-	I, D	Production	Commercial
SOAR +STEAM	R R, S	-	I, D	Production	Open source/ Commercial
MadKit	S	-	A, D	Production	Open Source

Table 1. Platform summary

Table 1 shows a summary overview of the presented agent platforms. A first thing to note is that some of the platforms, although historically pertaining to only one of the possible agent metaphors (M=middleware, R=reasoning, S=social) now start to address other areas as well, making them more generic and suitable for a wide range of application domains. JACK and SOAR, which started as pure reasoning platforms have been extended to support social concepts as well, and the Jadex platform presents an approach to integrate high-level reasoning with existing middleware technology.

Traditionally, only the middleware platforms are directly based on some existing or new standards. Some of them initially focused only on a single set of specifications (e.g. JADE, FIPA-OS), other such as ADK tried to provide support for a wide range of existing standards including FIPA, Web Services (WS) and others. Middleware support is a serious issue, as most newly developed applications have to be integrated with one or more existing systems. Although nowadays for most standards reusable third party libraries are available, when standards are not supported by the platform directly, the agent programmer has the tedious task of making the application to interoperate with other standards-compliant software.

In the recent years tool support has become more and more an issue for developers of agent platforms, but there is still some way to go until agent

technology is supported by development tools of the quality known from object oriented tools. None of the presented platforms provides all kinds of tools desirable for efficient application development (I=integrated development environments, A=administrative tools, D=debugging tools). For platforms supporting agents written in pure Java (here the middleware platforms and MadKit) existing Java-IDEs can still be used, with the advantage of a development environment already familiar to the programmer. In contrast newly developed IDEs (e.g. for the reasoning platforms) offer the advantage of directly supporting agent-oriented concepts.

Compared to the wide distribution of object oriented application frameworks (e.g. web containers or application servers) real case studies of applications developed with agent platforms are still scarce, or at least hard to find. Nevertheless, they do exist for most of the presented platforms, proving that successful agent applications can be built. Given that there is some 10-20 years gap between the first works on object oriented programming and the advent of the agent paradigm, it is reasonable to say that agent technology still has the potential to become as predominant as object oriented is at the moment.

Finally, the availability column shows if platforms are distributed as commercial products or open source implementations. Open source platforms are not only free in the sense that one does not have to pay for them, but also that there is the freedom to modify the platform itself, if needed. On the other hand, commercial products offer guaranteed support and should be mature and well tested. Some systems like SOAR are even available in both flavors. Therefore, different options are available for any kind of problem domain, and application developers can usually choose among a set of commercial products and open source implementations.

Although only a small cutout of available agent platforms has been presented, it should now be evident that there exists a large diversity in the different platforms. For example, no platform supports all three agent metaphors (middleware, reasoning, social). Many platforms claim a general applicability, but every platform is based on its own interpretation of the agent paradigm. Therefore, even though it might be somewhat usable in many domains, a platform would perform best in a domain where it offers a fitting agent metaphor, readily available tools and directly supported standards. Therefore, an agent developer carefully has to choose among the available options. In the following, the authors will try to give some guidelines how this choice can be simplified.

4 Guidelines for choosing among platforms and methodologies

One big problem of agent technology nowadays is its strong heterogeneity. This applies to the agent architectures (internal and social), to the methodologies and to the agent programming languages [PoBL2005b]. To further illustrate this issue one can look closer at the internal agent architecture BDI. Even though a consensus exits with respect to the basic concepts, the concrete interpretations and thus architectures and platforms differ considerably. In the field of agent oriented software engineering also a great variety of agent methodologies emerged. Some of them claim to be agent architecture independent such as Gaia [WoJK2000] whereas others are specifically tailored for some agent philosophy such as Prometheus [PaWi2004]. Although it might be tempting to use a generally applicable methodology it should be clear that such a methodology cannot support agent development with the same concepts as the platform does.

Carrying these considerations to the extreme, it is even contended how agent programming should be done. Some approaches favor new and specialized agent languages (e.g. JACK), whereas others employ existing programming languages such as Java (e.g. Jadex).

Hence, it becomes clear that the choice of the right combination of an agent methodology and a suitable platform is crucial for exploiting the potential of the agent paradigm for a given problem domain. This choice should start with an analysis of the problem domain gathering initial requirements and bringing to light the essential properties of the planned application. From these initial settings it should be discussed which agent philosophy deems most promising and allows the description of preferably many domain structures and behaviors. Having agreed on a common agent philosophy facilitates the selection of an agent platform and a suitable methodology considerably as it reduces the number of available candidates.

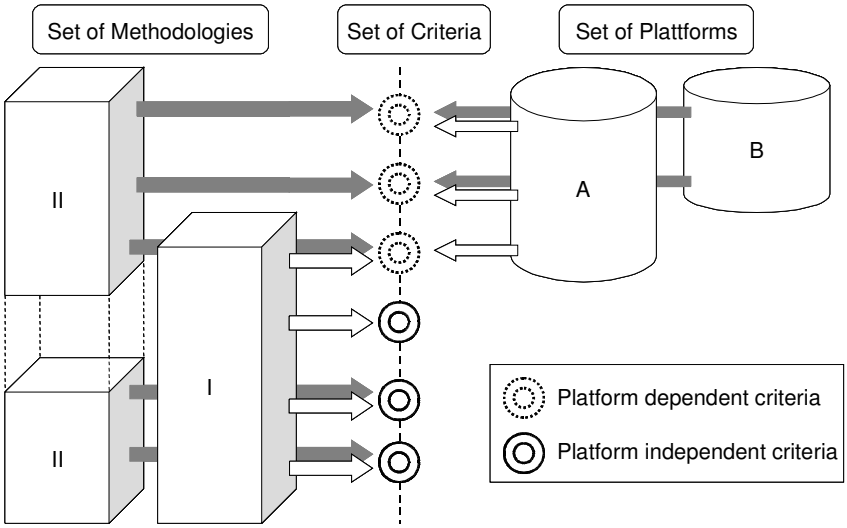

Figure 5. Selection framework

Given that a preselection of platforms and methodologies on basis of the favored agent philosophy has been carried out, the further selection process should not be done in isolation for either of both. Instead, it has to be found a constellation of methodology and platform that fits to each other.

For this process a general framework has been proposed in [SBPL2004]. It is based on a catalog of criteria that should be measured for both, the platform and the methodology candidates (cf. Figure 5). The set of criteria is divided into platform dependent and independent criteria whereby the independent criteria can be examined in a feature analysis. Categories for independent criteria include the notation (usability, expressiveness, etc.), the process (coverage of workflows, complexity, etc.) and pragmatic aspects (tool support, documentation, etc.). Independent criteria have been subject of several methodology comparisons that aimed to rank them with respect to the aforementioned factors [ShSt2001].

For platform dependent criteria (e.g. regarding the supported agent concepts) it needs to be determined if and how the methodology as well the platform supports a property. The match between them is analyzed to show their appropriateness. This means that a match with respect to a property exists when either platform and methodology support a considered property in the same (or a very similar) way or when both do not support the property. The shared absence of a property is regarded as a match here, because the absence of a concept in both platform and methodology also identifies appropriateness.

To arrive at a final decision, the platform dependent criteria should be weighted according to the demands of the application domain as some agent concepts (e.g. mobility) might be irrelevant for a given domain. For each pair of methodology and platform the overall match quality can be estimated. The platform and methodology pair with the weighted best match should be chosen. This selection process can be simplified if the preselection phase is rather rigid or if for some external reasons (e.g. company relationships) a certain platform or methodology has to be used.

5 Conclusions

This chapter has presented an overview of agent standards and platforms. The agent platforms have been categorized by their main architectural focus leading to three different classes: middleware, reasoning and social platforms. Middleware platforms address primarily layers L1-L3 of the reference architecture focusing on support for interoperability with other FIPA compliant platforms. Secondly, reasoning platforms mainly deal with the agent internal decision process that leads to concrete agent behavior. Thirdly, social platforms highlight organizational structures as well as coordinated (team) behavior. Based on the criteria *Concepts*, *Standards*, *Tools*, and *Applications* typical representatives of the respective categories have been evaluated.

Given that a vast amount of different platforms belonging to one or another category exists, this chapter also sketches a systematical approach for choosing a platform for a specific development project. Roughly speaking, the approach consists of two phases; a domain dependent preselection phase and a subsequent stage for platform/methodology evaluation. In the first stage a domain analysis is used to set-up a fitting agent metaphor emphasizing the important aspects of the domain. Thereafter, in the second stage the remaining platforms are evaluated together with possibly fitting methodologies. The basic assumption in this connection is that an agent platform and a concomitant methodology are strongly interrelated and should be chosen together for guaranteeing effective application development. As a result, one obtains estimated quality measurements for platform-methodology pairs. The pair exhibiting the best match and the best criteria coverage should be chosen.

Further reading

[UnCK2005] Unland, R.; Calisti, M.; Klusch, M.: Software Agent-Based Applications, Platforms and Development Kits. Birkhäuser, 2005.

[BDDE2005] Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programing Multi-Agent Systems. Kluwer Academic Publishers, 2005.

[PoBL2005b] Pokahr, A.; Braubach, L.; Lamersdorf, W.: Agenten: Technologie für den Mainstream? In: it – Information Technology 5(2005), pp.300-307.

[SBPL2004] Sudeikat, J.; Braubach, L.; Pokahr, A.; Lamersdorf, W.: Evaluation of Agent-Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform. In: Giorgini, P.; Müller, J.P.; Odell, J. (Eds.): Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004. Springer Verlag, 2004, pp. 126-141.

[WeJa2004] Weiß, G.; Jakob, R.: Agentenorientierte Softwareentwicklung – Methoden und Tools. Xpert.press Reihe, Springer-Verlag, September 2004. ISBN 3-540-00062-3.

References

[AmSt1988] Ambros-Ingerson, J.; Steel, S.: Integrating Planning, Execution and Monitoring. In: Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88); St. Paul, MN. AAAI Press, Menlo Park, CA, 1988, pp. 83-88.

[BaMa1999] Baeumer, C.; Magedanz, T.: Grasshopper: A Mobile Agent Platform for Active Telecommunication Networks. In: Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA-99). Springer, Berlin, 1999, pp. 19-32.

[BBCP2005] Bellifemine, F.; Bergenti, F.; Caire, G.; Poggi, A.: JADE - A Java Agent Development Framework. In: Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programing Multi-Agent Systems. Kluwer Academic Publishers, 2005.

[BDDE2005] Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programing Multi-Agent Systems. Kluwer Academic Publishers, 2005.

[BoHV2005] Bordini, R.; Hübner, J.; Vieira, R.: Jason and the Golden Fleece of Agent-Oriented Programming. In: Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programing Multi-Agent Systems. Kluwer Academic Publishers, 2005.

[BPML2004] Braubach, L.; Pokahr, A.; Moldt, D.; Lamersdorf, W.: Goal Representation for BDI Agent Systems. In: Bordini, R. et al.

(Eds.): Proceedings of the 2nd International Workshop on Programming Multiagent Systems, Languages and Tools (PROMAS 2004), 3rd International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'04), New York, USA. Lecture Notes in Computer Science. Springer-Verlag, Berlin, New York, 2005, pp. 46-67.

[Brat1987] Bratman, M.: Intention, Plans, and Practical Reason. Harvard University Press, 1987.

[BrIP1988] Bratman, M.; Israel, D.; Pollack, M.: Plans and Resource-Bounded Practical Reasoning. In: Computational Intelligence 4(1988)4, pp. 349-355.

[Broo1986] Brooks, R.: A Robust Layered Control System for a Mobile Robot. In: IEEE Journal of Robotics and Automation 2(1986)1, pp. 24-30.

[BrPL2005] Braubach, L.; Pokahr, A.; Lamersdorf, W.: Jadex: A BDI Agent System Combining Middleware and Reasoning. In: Unland, R.; Calisti, M.; Klusch, M. (Eds.): Software Agent-Based Applications, Platforms and Development Kits. Birkhäuser, 2005.

[BHRH2000] Busetta, P.; Howden, N.; Rönnquist, R.; Hodgson, A.: Structuring BDI Agents in Functional Clusters. In: Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL'99), LNCS 1757. Springer, 2000, pp. 277-289.

[CFBB2004] Calisti, M.; Funk, P.; Biellman, S.; Bugnon, T.: A Multi-Agent System for Organ Transplant Management. In: Applications of Software Agent Technology in the Health Care Domain. Springer, Heidelberg, 2004.

[ClPE1997] Clement, P.; Papaioannou, T.; Edwards, J.: Aglets: Enabling the Virtual Enterprise. In: Proceedings of the International Conference Managing Enterprises – Stakeholders, Engineering, Logistics and Achievement' (ME-SELA '97), 1997.

[CoLe1991] Cohen, P.; Levesque, H.: Teamwork, SRI International. Technote 504. Menlo Park, CA, 1991.

[Denn1987] Dennett, D.: The Intentional Stance. Bradford Books, 1987.

[EiMa2002] Eiter, T.; Mascardi, V.: Comparing Environments for Developing Software Agents. In: AI Communications 15(4), pp. 169-197, 2002.

[Ferb2003] Ferber, J.: From Agents to Organizations: An Organizational View of Multi-Agent Systems. In: Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, 2003. Revised Papers, 2003, pp. 214-230.

[GrKr1996] Grosz, B.; Kraus, S.: Collaborative Plans for Complex Group Action. In: Artificial Intelligence 86(1996)2, pp. 269-357.

[GuFe2001] Gutknecht, O.; Ferber, J.: The MADKIT Agent Platform Architecture. In: Wagner, T.; Rana, O. (Eds.): Revised Papers From the International Workshop on Infrastructure for Multi-Agent

Systems: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, June 3-7, 2000. Lecture Notes in Computer Science 1887(2001). Springer-Verlag, London, 2001, pp. 48-55.

[HoRB1999] Hodgson, A.; Rönnquist, R.; Busetta, P.: Specification of Coordinated Agent Behavior (The SimpleTeam Approach). In: Proceedings of the Workshop First International Conference on Team Behaviour and Plan Recognition at IJCAI-99. Stockholm, Sweden, 1999.

[HRHL2001] Howden, N.; Rönnquist, R.; Hodgson, A.; Lucas, A.: JACK Intelligent Agents – Summary of an Agent Infrastructure. In: Proceedings of the 5th ACM International Conference on Autonomous Agents. Canada, 2001.

[JeMa1992] Jennings, N.; Mamdani, E.: Using Joint Responsibility to Coordinate Collaborative Problem Solving in Dynamic Environments. In: AAAI. 1992, pp. 269-275.

[LeLR1996] Lehman, J.; Laird, J.; Rosenbloom, P.: A gentle introduction to Soar, an architecture for human cognition. In: Invitation to Cognitive Science 4(1996), MIT Press.

[MaDA2005] Mascardi, V.; Demergasso, D.; Ancona, D.: Languages for Programming BDI-style Agents: an Overview. In: Corradini, F.; De Paoli, F.; Merelli, E.; Omicini, A. (Eds.): Proceedings of WOA 2005 dagli Oggetti agli Agenti Simulazione e Analisi Formale di Sistemi Complessi Pitagora Editrice Bologna, ISBN 88-371-1590-3. pp. 9-15.

[Mang2002] Mangina, E.: Review of Software Products for Multi-Agent Systems. In: AgentLink, software report, 2002.

[Mil+1998] Milojicic, D.; Breugst, M.; Busse, I.; Campbell, J.; Covaci, S.; Friedman, B.; Kosaka, K.; Lange, D.; Ono, K.; Oshima, M.; Tham, C.; Virdhagriswaran, S.; White, J.: MASIF: The OMG Mobile Agent System Interoperability Facility. In: Proceedings of the International Workshop on Mobile Agents (MA'98), 1998.

[Newe1990] Newell, A.: Unified Theories of Cognition. Harvard University Press, 1990.

[Norl2004] Norling, E.: Folk Psychology for Human Modelling: Extending the BDI Paradigm. In: Proceedings of in the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 2004.

[OdPF2003] Odell, J.; Parunak, H.; Fleischer, M.: The Role of Roles in Designing Effective Agent Organizations. In: Software Engineering for Large-Scale MAS. Springer, Heidelberg, 2003, pp. 27-38.

[OMG2000] Object Management Group (OMG): Mobile Agent Facility Specification. http://www.omg.org/cgi-bin/doc?formal/2000-01-02, 2000.

[PaWi2004] Padgham, L.; Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons, New York, 2004.

[PoBL2005a] Pokahr, A.; Braubach, L.; Lamersdorf, W.: Jadex: A BDI Reasoning Engine. In: Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programing Multi-Agent Systems. Kluwer Academic Publishers, 2005, pp. 149-174.

[PoBL2005b] Pokahr, A.; Braubach, L.; Lamersdorf, W.: Agenten: Technologie für den Mainstream? In: it – Information Technology 5(2005), pp.300-307.

[PoCh2001] Poslad, S.; Charlton, P.: Standardizing Agent Interoperability: The FIPA Approach. In: 9th ECCAI Advanced Course, ACAI 2001 and Agent Links 3rd European Agent Systems Summer School, EASSS 2001, Prague, Czech Republic. Springer, Heidelberg, 2001.

[RaGe1995] Rao, A.; Georgeff, M.: BDI agents: From theory to practice. In: Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95). San Francisco, CA, USA, 1995, pp. 312-319.

[Rao1996] Rao, A.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: van der Velde, W.; Perram, J. (Eds.): Agents Breaking Away. Springer, Berlin, Heidelberg, New York, 1996.

[SBPL2004] Sudeikat, J.; Braubach, L.; Pokahr, A.; Lamersdorf, W.: Evaluation of Agent-Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform. In: Giorgini, P.; Müller, J. P.; Odell, J. (Eds.): Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004. Springer Verlag, 2004, pp. 126-141.

[Sear1969] Searle, J. R.: Speech Acts: an essay in the philosophy of language. Cambridge University Press, 1969.

[ShSt2001] Shehory, O.; Sturm, A.: Evaluation of modeling techniques for agent-based systems. In: Proceedings of the fifth international conference on Autonomous agents (Agents 2001). ACM, Montreal, Canada, 2001, pp. 624-631.

[Tamb1997] Tambe, M.: Towards Flexible Teamwork. In: Journal of Artificial Intelligence Research 7(1997), pp. 83-124.

[Weis2002] Weiß, G.: Agent Orientation in Software Engineering. In: Knowledge Engineering Review 16(2002)4, pp. 349-373.

[WiCR2002] Willmott, S.; Calisti, M.; Rollon, E.: Challenges in Large-Scale Open Agent Mediated Economies. In: Proceedings of AAMAS '02: Revised Papers from the Workshop on Agent Mediated Electronic Commerce on Agent-Mediated Electronic Commerce IV, Designing Mechanisms and Systems. Springer, Berlin, Heidelberg, New York, 2002.

[Wini2005] Winikoff, M.: JACK Intelligent Agents: An Industrial Strength Platform. In: Bordini, R.; Dastani, M.; Dix, J.; El Fallah Seghrouchni, A. (Eds.): Programing Multi-Agent Systems. Kluwer Academic Publishers, 2005, pp.175-193.

[WoJe1995] Wooldridge, M.; Jennings, N.: Intelligent Agents: Theory and Practice. In: The Knowledge Engineering Review 10(1995)2, pp. 115-152.

[WoJK2000] Wooldridge, M.; Jennings, N.; Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. In: Autonomous Agents and Multi-Agent Systems 3(2000)3, pp. 285-312.