EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Agent Technology and its Applications

Učební texty k semináři

Autoři:

doc. Dr. Ing. Michal Pěchouček, M.Sc. (ČVUT v Praze)
Ing. Michal Jakob, Ph.D. (ČVUT v Praze)
Dr. rer. nat. Peter Novák (ČVUT v Praze)
Ing. Martin Rehák, Ph.D. (ČVUT v Praze)
Ing. David Šišlák (ČVUT v Praze)
Ing. Jiří Vokřínek (ČVUT v Praze)

Datum:

22. března 2010

Centrum pro rozvoj výzkumu pokročilých řídicích a senzorických technologií
CZ.1.07/2.3.00/09.0031

TENTO STUDIJNÍ MATERIÁL JE SPOLUFINANCOVÁN EVROPSKÝM SOCIÁLNÍM FONDEM A STÁTNÍM ROZPOČTEM ČESKÉ REPUBLIKY

Contents

Chapter 1

Introduction to Agent-based Computing and Multi-agent Systems

Agent-based computing is a subfield of computer science and artificial intelligence, that studies the concepts of autonomy of individual computational processes (running either software applications or hardware robots) and interaction between such heterogeneous autonomous process. Agent-based computing leverages results from mathematical logics, game theory, mechanism design, machine learning, automated planning and others. Multiagent systems are collections of autonomous agents, which either simulates or control distributed cooperative/competitive systems. Agent technology is a collection of methods, algorithms and software tools that support development of mutliagent systems.

The research field of **autonomous agents and multi-agent systems** (also referred to as **agent-based computing**), a specific sub-field of computer science and artificial intelligence, investigates the concepts of autonomous decision making, communication and coordination, distributed planning and distributed learning but also game-theoretic aspects of competitive behaviour or logical formalization of higher level knowledge structures representing interaction attitude of actors in multi-actor environment. **Agent technology** provides a set of tools, algorithms and methodologies for development of distributed, asynchronous intelligent software applications that leverage the above listed theories.

As the society is moving closer to the network era linking many users, hardware assets and hosted computational process, there is a tremendous application potential of this research field. E.g. adaptive, real-time coordination capability in robotics is very important for aerial vehicles providing surveillance and search in rescue operations, natural disasters (like fires) or security applications (providing live feed of larger protected areas). Multi-agent algorithms can also provide novel mechanisms for specialized applications in air-traffic control, large cities traffic modelling and planning or intelligent building design. These applications support or control the interaction of large number of non-trivial interacting entities (be it air planes, cars, or sensors in the building). Agent-based computing can also support peer-to-peer knowledge and data sharing in social networks, by allowing intelligent agents

to negotiate various data confidentiality policies, and thus support wider exploitation of knowledge and experience. Adoption of agent-based techniques in these domains not only facilitates control of distributed systems, but also provide scalability, robustness and reliability, which cannot be achieved by centralized control. Even though there has been an early deployment of agents in industry, a wider adoption of this novel and innovative concept remains a great challenge for researchers and early adopters.

A **multi-agent system** is a decentralized computational (software) system, often distributed (or at least open to distribution across hardware platforms) whose behavior is defined and implemented by means of complex, peer-to-peer interaction among autonomous, rational and deliberative units – agents.

An **agent** is an encapsulated computational (or physical, even human) system, that is situated in some environment, and that is capable of flexible, autonomous behavior in order to meet its design objective [48]. The agent can exists on its own but often is a component of a multi-agent system.

There are the following key properties of an autonomous intelligent agent:

- *Autonomy* – the agent is accountable for execution of its own actions and is not controlled from outside. Often the agent's reasoning mechanism that selects the action to be executed is unknown from outside of the agent (unlike e.g. objects).

- *Reactivity* – the agent is able react qucikly to the events in the environment and to the requests from other agents, it is able to reconsider her activity according to the change of the environment in timely fashion. Often the longest reasoning cycle of an agent needs to perform faster than the fastest change in the environment

- *Intentionality* – the agents is able to maintain her long term intention encoded by the agent's designer and is capable to consider both the long term intentions and immediate reactive inputs when selecting the next action.

- *Social capability* – the agent is able to interact, collaborate, form teams but also to perferm different levels of reasoning about the other agents.

The concept of agents has been successfully applied in the three following ways: (i) *agents as design metaphore*, providing the designers and developers with a way of structuring an application around autonomous, communicative elements; (ii) *source of technologies*, providing specific techniques and algorithms for dealing with interactions in dynamic and open environments and (iii) *simulation models*, providing strong models for representing complex

and dynamic real-world environments. The concept of agents is in either of the use models considered at the following design levels:

- *organization-level*: related to the agent communities as a whole (organizational structure, trust, norms, obligations, self-organisation, etc.);
- *interaction-level*: concern communication among agents (languages, interaction protocols, negotiations, resource allocation mechanisms);
- *agent-level*: concern individual agents (agent architecture, reasoning, learning, local processing of social knowledge).

1.1 Formal Models of Agent Based Systems

In the following we will introduce the concept of agents formally. Let us have possible states of the environment: $S = \{s, s', \dots\}$; possible actions that can transform the environment:
$\mathcal{A}c = \{\alpha, \alpha', \dots\}$; simple model of the environment $\tau : S \times \mathcal{A}c \to \wp(S)$ and B, G as a set of agents beliefs and goals respectively. In such a model the *stateless* purely, reactive agents are specified by a function with a signature as $\mathcal{A}g : S \to \mathcal{A}c$, the *stateful* agents are defined as $\mathcal{A}g : S^* \to \mathcal{A}c$, the *goal-directed*, *knowledgeable* agents are defined as $\mathcal{A}g : B \times G \to \mathcal{A}c$; and the *autonomous* agents is defined as $\mathcal{A}g : S^* \to \wp(\mathcal{A}c)$.
Given the *model of the history* of an agent operation A in the environment

$$r : \{s_0 \xrightarrow[\alpha_0]{} s_1 \xrightarrow[\alpha_1]{} s_2 \xrightarrow[\alpha_2]{} \dots \xrightarrow[\alpha_3]{} s_n\}$$

we introduce $\mathcal{R}$ – **characteristic behavior** – as the set of all finite possible histories of agents behaviour.
The environment in which the agent operates is either

- deterministic, history independent: $\tau : S \times \mathcal{A}g \to S$
- non-deterministic, history independent: $\tau : S \times \mathcal{A}g \to \wp(S)$
- history dependent environment $\mathcal{E}_{nv} = \langle S, s_0, \tau \rangle$
  - $\mathcal{R}^{\mathcal{A}c} \subseteq \mathcal{R}$ so that it finishes with an action,
  - $\mathcal{R}^{\mathcal{E}_{nv}} \subseteq \mathcal{R}$ so that it finishes with an environment state

$$\tau : \mathcal{R}^{\mathcal{A}c} \to \wp(S)$$

- model of agents that inhabit the system:

$$\mathcal{A}g : \mathcal{R}^{\mathcal{E}_{nv}} \to \wp(\mathcal{A}c)$$

The characteristic behavior $\mathcal{R}$ of an agent $\mathcal{A}g$ in an environment $\mathcal{E}_{nv}$ is $\mathcal{R}(\mathcal{A}g, \mathcal{E}_{nv})$. If we consider only *finite histories*, the sequence $r : \{s_0 \overrightarrow{\alpha_0} s_1 \overrightarrow{\alpha_1} s_2 \overrightarrow{\alpha_2} \ldots \overrightarrow{\alpha_3} s_n\}$ represents a behavior of an agent $\mathcal{A}g$ in an environment $\mathcal{E}_{nv}$ provided that:

- s_0 is an initial state of $\mathcal{E}_{nv}$
- $\alpha_0 \in \mathcal{A}g(s_0)$
- $\forall n > 0$:

  $s_n \in \tau((s_0, \alpha_0, \ldots s_{n-1}, \alpha_{n-1}))$ and
  $\alpha_n = \mathcal{A}g((s_0, \alpha_0, \ldots s_{n-1}, \alpha_{n-1}, s_n))$

Two agents are behaviorally equivalent iff $\mathcal{R}(\mathcal{A}g_1, \mathcal{E}_{nv}) = \mathcal{R}(\mathcal{A}g_2, \mathcal{E}_{nv})$
Rational agents are motivated to optimize their utility functions. Agent's short-term utility function $u : S \rightarrow \mathbb{R}$ labels the agent's individual states - denoting how profitable is it for the agent to bring about the particular state. Agent's long term utility function $--$ $u : \mathcal{R} \rightarrow \mathbb{R}$ assigns a utility not to the individual states, but to the specific runs. The probability of a run to happen is defined as follows:

$$P(r|\mathcal{A}g, \mathcal{E}_{nv}) \sum_{\forall r : r \in \mathcal{R}(\mathcal{A}g, \mathcal{E}_{nv})} P(r|\mathcal{A}g, \mathcal{E}_{nv}) = 1$$

Based on the definitions above, the rational agent is maximizing its **expected utility** $u(r)P(r|\mathcal{A}g, \mathcal{E}_{nv})$ – a measure that represents a probability of all individual runs that can happen and the profit they may provide:

$$\mathcal{A}g_{\texttt{rational}} = \arg \max_{\mathcal{A}g \in \mathcal{AG}} \sum_{\forall r : r \in \mathcal{R}(\mathcal{A}g, \mathcal{E}_{nv})} u(r)P(r|\mathcal{A}g, \mathcal{E}_{nv}).$$

Rationality is a complex concept and can be made more specific in restricted scenarios. Let us define the concept of **bounded rationality** as

- $\mathcal{AG}_m = \{\mathcal{A}g | \mathcal{A}g \in \mathcal{AG}$ and $\mathcal{A}g$ can be implemented on machine $m\}$
- $\mathcal{AG}_p = \{\mathcal{A}g | \mathcal{A}g \in \mathcal{AG}$ and $\mathcal{A}g$ can be implemented so that for any input it gives polynomially bounded output$\}$

and the concept of **calculative rationality** as

- $\mathcal{AG}_{CR} = \{\mathcal{A}g | \mathcal{A}g \in \mathcal{AG}$ and $\mathcal{A}g$ can be implemented so that for any reaction in the environment it provides an output faster than that the environment changes again$\}$.

The performance objective of the given agent is given by a task that is formally described as **predicate task specification** $\Psi : \mathcal{R} \rightarrow \{0, 1\}$. Given a task environment $\langle \mathcal{E}_{nv}, \Psi \rangle$ we want to build agents that provide characteristic behavior defined as:

$$\mathcal{R}_{\Psi}(\mathcal{A}g, \mathcal{E}_{nv}) = \{r | r \in \mathcal{R}(\mathcal{A}g, \mathcal{E}_{nv}) \wedge \Psi(r)\}$$

Probability of an agent accomplishing successfully the task Ψ is defined as:

$$P(\Psi | \mathcal{A}g, \mathcal{E}_{nv}) = \sum_{r=\mathcal{R}_{\Psi}(\mathcal{A}g, \mathcal{E}_{nv})} P(r | \mathcal{A}g, \mathcal{E}_{nv}).$$

Success in task completion can be either *pessimistic* $\forall r \in \mathcal{R}(\mathcal{A}g, \mathcal{E}_{nv}) : \Psi(r)$ or *optimistic* $\exists r \in \mathcal{R}(\mathcal{A}g, \mathcal{E}_{nv}) : \Psi(r)$. While the pessimistic definition defines the *maintenance task*, the optimistic definition defines the *achievement task*.

1.2 Agent Architectures

1.2.1 Reactive Agents

Reactive agents represent the simplest possible form of agency. The key idea of the reactive architecture is the computational implementation of **reactive intelligence**. In reactive intelligence the rational decision making is closely linked to the environment. Intelligent behavior of reactive agents emerges from the interaction of various simpler (usually rule-based) behaviors. Philosophy of reactive agents rejects the classical AI approach of implementing decision making by means of symbolic knowledge representation and inference.

This is why the reactive agents contain no symbolic knowledge representation (ie: no state, no representation of the environment, no representation of the other agents, ...). Their behavior is defined by a set (may be large) of `perception-action` rules.

The best known reference architecture for the reactive agents is the **subsumption architecture**, introduced by Brooks [38]. Brooks introduced the concept of *task accomplishing behavior* – a function (implemented by a stateless rule) that maps a `percept` into an `action`. The behaviors are structured into *subsumption hierarchy* according to the different priority of the respective rules.

An alternative approach – **situated automata** – has been suggested by [19]. In this paradigm, a behavior of an agent is specified in a declarative language (rather more expressive than rules), and this specification is compiled down

into a rule-based agent, which satisfies the declarative specification. This agent can operate in a provable time bound. The important part of reasoning is done off line, at the compile time, rather than online at run time. The limitations of the approach are given by the fact the compilation process is equivalent to an NP-complete problem. Maes suggested the **agent network architecture** where an agent is a set of competence modules that loosely resemble the Brooks (hence reactive) behaviors. Each module is specified by preconditions, postconditions and an activation level (that gives relevance of the respective rule).

1.2.2 Deliberative Agents

The deliberative agents are such agents that (i) contain an explicitly represented symbolic model of the world, the problem, their knowledge, their history and (ii) make decisions (e.g. about what actions to perform) by means of symbolic reasoning.
When design a deliberative agent we must tackle:

- **transduction problem**: that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful (e.g. vision, speech understanding, learning),
- **representation problem**: that of how to symbolically represent information about complex real-world entities and processes and
- **reasoning problem**: how to get agents to reason with this information in time for the results to be useful (e.g. knowledge representation, automated reasoning, automatic planning).

The reasoning process of the deliberative agents can be implemented by *theoretically reasoning* (e.g. deductive agents) or by *practically reasoning* (e.g. BDI agents).

Deductive Agents. The deductive agents use the logical deduction and theorem proving as a reasoning model in order to manipulate the symbolic representation encoded in the form of logical formulae[1].
The deductive reasoning can be designed and used in two possible ways: (i) either there is a constantly running reasoning loop that is trying to prove that there is an action that shall, or at least is allowed to be carried out, or

[1]Logic can be used in agency twofold: (i) for implementing agents' reasoning mechanism and (ii) for agents' behavior, decision making, social behavior specification

(ii) the mechanism tries to prove what is the best reaction to a new piece of information sensed from the environment.
The deductive reasoning can be designed used twofold: (i) either there is a constantly running reasoning loop that is try to prove that there is an action that shall, or at least is allowed to be carried out – given agent' s *knowledge* Δ, and its *goal* $\mathcal{G}$ we shall instantiate a variable α when proving ${}_{\Delta,\mathcal{G}} \vdash \exists\alpha$ $\texttt{shall-be-done}(\alpha) \vee \texttt{may-be-done}(\alpha) \Rightarrow \texttt{do}(\alpha)$, or (ii) the mechanism tries to prove what is the best reaction to a new piece of information $\mathcal{P}$ sensed from the environment ${}_{\Delta,\mathcal{G}} \vdash \exists\alpha\ \texttt{react}(\mathcal{P}, \alpha)$
The concept of deductive reasoning provides a very expressive mechanism for designing a wide range of complex behavioral patterns. However, proving an appropriateness or plausibility of an action to be carried out may take longer that the calculative rationality (as defined in [48]) assumption requires.

Practical Reasoning (BDI) Agents. Unlike deductive reasoning that is a computational activity oriented towards a true facts, practical reasoning is a decision process oriented towards an action (a process of figuring out what to do).
Practical reasoning consists of two activities: (i) **deliberation**, deciding *which state* we want to achieve and (ii) **means-ends reasoning**, deciding *how to achieve* this state.
Based on theory of practical reasoning Rao and Georgeff' [33] formulated a Belief-Desire-Intention (BDI) model of agency as a framework for reasoning about formal abstract models of agents' mental states. Besides implementing deliberation and means-end reasoning as the key reasoning processes, the BDI agent organizes its knowledge into the following knowledge structures:

- **beliefs**, which constitute agent's knowledge of the state of the environment, agent's internal state, knowledge about the other agents,
- **desires**, which determine agent's long-term motivation, what is the agent trying to bring about, maintain, achieve, etc., and
- **intentions**, which capture agent's real-time decisions about how to act in a particular situation in order to fulfill the given desires.

The intentions link the deliberation process and the means-end reasoning process. Unlike a desire, an intention may be seen as agents' immediate commitment to achieve a specific state of affairs that drives the means-end reasoning process to identify the right action (or a series of action) that result in the given state of affairs.

1.3 Research Challenges and Applications of Agent Based Computing

Currently the research community of autonomous agents and multi-agent systems is addressing a wide range of complex research challenges aiming at providing the following capabilities:

- *Coordination* - list of agent techniques (based mainly on dedicated coordination protocols and various collaboration enforcement mechanisms) that facilitates coordinated behavior between autonomous, while collaborative agents. Coordination usually supports conflict resolution and collision avoidance, resource sharing, plan merging, and various collective kinds of behavior.

- *Negotiation* - list of various negotiation and auctioning techniques that facilitate an agreement about a joint decision among several self-interested actors or agents. Here we emphasize mainly negotiation protocols and mechanisms how individual actors shall act and what strategies shall they impose to optimize their individual utility.

- *Simulation* - techniques that allow inspection of collective behavior of the interactive actors, provided that the models of the individual agents are known. Here we count on the versatile simulation frameworks that allow long-run complex simulation and various "what-if" analyses of different problems. If distributed hardware system is modeled, agent-based simulation enables a close linkage between simulation and the real hardware machinery.

- *Interoperability* - set of techniques for achieving high level interoperability among software components developed by different designers, especially in the situation where the source code and complete models of behavior are not shared. Interoperability is studied on the level of physical connections via interaction protocols but also semantics of communication.

- *Adjustable Autonomy and Policies* - set of techniques and methods for specifying and dynamic adjustment of decision making autonomy of the individual actors in a multi-agent system. Various formal frameworks for specifying policies have been proposed and numerous policy management systems have been designed by the agent community.

- *Organization* - techniques supporting agents in ability to organize autonomously in permanent or temporal interaction and collaboration struc-

tures (virtual organizations), assign roles, establish and follow norms, or comply with electronic institutions.

- *Multi-agent Learning* - in the multi-agent community there are various methods allowing an agent to form hypothesis about available agents. These methods work mainly with the logs of communication or past behavior of agents. Agent community also provides techniques for collaborative (distributed) learning, where agents may share learnt hypothesis or observed data. A typical application domain is distributed diagnostics.

- *Multi-agent Planning* - specific methods of collaboration and sharing information while planning operation among autonomous collaborating agents. Agent community provides methods for knowledge sharing, negotiation and collaboration during the 5 phases of distributed planning (Durfee, 1999): task decomposition, resource allocation, conflict resolution, individual planning, and plan merging. These methods are particularly suitable for the situations when the knowledge needed for planning is not available centrally.

- *Knowledge Sharing* - techniques assisting in sharing knowledge and understanding different types of knowledge among collaborative parties as well as methods allowing partial knowledge sharing in semi-trusted agent communities (Pechouček, et. al. 2002) (closely linked with distributed learning and distributed planning).

- *Trust and Reputation* - methods allowing each agent to build a trust model and share reputation information about agents. Trust and reputation is used in non-collaborative scenario where agents may perform non-trusted and deceptive behavior.

When analyzing the opportunities for agent technology deployment it is fair to distinguish between two slightly different sets of techniques: (i) techniques supporting interaction and collaboration of distributed multiple agents and (ii) techniques supporting agents' autonomy. Even though the combination of both aspects of agency is a desirable property of an agent-based system, in our experience industrial deployment emphasize either the distributed and collective aspects or the autonomy-oriented aspects of agency. Let us discuss the properties of the problems and the application requirements with respect to what the agent techniques can provide.

Distributed and collective aspects of agency are considered to perform well in application domains with the following specific properties (properties P1 - P6):

- **Decentralized scenarios:** Particularly suitable are the domains where the data and knowledge required for computation are not or cannot be available centrally or the process physical system control needs to be distributed. This can be the case in several situations (properties P1 - P3):
  - *Geographical distribution* of knowledge and control (e.g., logistics, collaborative exploration, mobile and collective robotics, pervasive systems) or the environments with partial or temporary communication inaccessibility (where self-organization, local interaction and intelligent synchronization is needed in order to cope with communication inaccessibility) - property P1.
  - *Competitive domains*, with the restrictions on the information sharing (e.g., e-commerce applications, supply-chain management, and e-business) - property P2.
  - Domains with the requirements for *time-critical response* and *high robustness* in distributed environment (e.g., time-critical (soft- and/or hard-realtime) manufacturing or industrial systems control, with replanning, or fast local reconfiguration) - property P3.
- **Simulation and modeling scenarios:** Using agents for simulation purposes has been very common, while the right justification was often missing. Agents shall be deployed in simulation exercises where we require, e.g., an easy migration from the simulation to deployment in real environment - property P4.
- **Open systems scenarios:** In scenarios requiring integration and interoperability among software systems that are not known a priori and whose source code may not be available - here the use of agent technologies, especially agent communication languages and interoperability standards is advisable - property P5.
- **Complex systems scenarios:** In scenarios requiring modeling, controlling or engineering of complex systems. Decomposition of the decision making into separate agents' reasoning and solving problems by means of negotiation represents a novel software development paradigm (Giorgini, et.al. 2003). Complex system modeling is often understood as closely related to solving complex problems. Potentials for decreasing computational requirements for complex problem solving by means of paralleling the computational process within multiple agents is limited, but possible - property P6.

Autonomy oriented aspects of agency is appropriate in application domains with high requirements for systems with decision making autonomy, when the user delegates the substantial amount of decision making authority to the system and when the system is expected to cope independently with unexpected situations (also in the situation with long term communication inaccessibility and interaction isolation of the autonomous entity) - property P7.
The properties P1 and P7 are somewhat linked. In the situations, where the communication infrastructure, a critical component required for collective decision making, is disrupted, some of the agents need to perform higher level of decision making autonomy.

Chapter 2

Introduction to Programming Autonomous Agents and Multi-agent Systems

Motivation

Intelligent agents are assumed to be *autonomous*, *proactive*, *reactive*, as well as *socially able* (cf. [48]). In this tutorial, we are concerned with the problem of *programming cognitive agents* (also *knowledge-intensive agents* [18]). I.e., those employing cognitive processes as the basis for their decision making and actions. In particular, we will focus on systems constructing and maintaining a *mental state* [42] which is used as the basis for the action selection. Being able to process symbolic information is a necessary prerequisite for inter-agent information exchange. Reasoning and communication about agent's mental attitudes, such as *beliefs*, *goals*, *intentions*, *commitments*, *obligations*, etc. facilitates coordination among agents. In consequence, programming frameworks for such a agents must support *programming with mental attitudes*.

Foundations: BDI Architecture

Behaviour of embodied agents can be described by an *agent program*, a mapping of agent's percepts to actions ([41]). The straightforward encoding in terms of a table associating actions with perceptions leads to reactive agent programs. While useful for relatively small and isolated systems, when dealing with rich environments such programs become brittle, if not impossible to construct. This programming style tackles the issue of agent reactivity, however does not facilitate implementation of proactive and socially able systems.

On the other hand, approaches based on classical planning support more compact agent programs, however blind plan execution suffers in highly dynamic and uncertain environments. High rate of environment changes leads to frequent invalidation of agent's plans. In turn, in applications with costly re-planning, such an approach leads to ineffective use of agents resources and potentially completely disables agent's behaviour.

To tackle these problems, the research led to development of *hybrid architectures*. As one of the most useful and in the recent years widely applied, emerged the *Belief-Desire-Intention* (BDI) abstract architecture by Rao and Georgeff [34, 35]. Firstly, it prescribes decomposition of agent's mental attitudes into three distinct categories: beliefs, desires and intentions of an agent. Secondly, it comes with an abstract interpreter and an associated logic-based formalism (CTL*) facilitating reasoning about properties of evolutions of such systems. It prescribes a reasoning model (*I-System*) for BDI-style rational agents in terms of the following eight axioms: Informally, an agent should adopt only goals it believes to be an option (AI1). It should adopt intentions only in order to achieve its goals (AI2). If the agent has an intention to perform a certain action, it will eventually also perform it (AI3). It should be aware of the fact that it committed itself to certain goals and intentions (AI4, AI5). If the agent intends to achieve something, it also has to have a goal to intend it (AI6). It should be aware of its actions (AI7) and finally the agent should never hold its intentions forever (AI8).

Agent-oriented Programming Languages

The abstract BDI architecture led to agent programming techniques employing reactive planning techniques while not necessarily performing proper planning. The resulting programs take a form of a set of partial plans enacted according to the current state of agent's mental state.
Attempts to develop operationalization of the proposed abstract BDI architecture [34, 35] married with the notion of agent oriented programming (AOP) [42] resulted in rise of *agent-oriented programming languages*. Historically, the most important include logic-based approaches AGENT-0 [42], PLACA [44], AgentSpeak(L)/Jason [36, 5], Golog [21], as well as more recent proposals 3APL [11], GOAL [13], MetateM [12] and more pragmatically oriented proposals such as e.g., JACK [47], or Jadex [29].
In the tutorial, we introduce AgentSpeak(L)/Jason and Jadex in a more detail. While the former belongs to the group of approaches based on a formal logic-based theory, the latter is a more pragmatic attempt to bring the AOP techniques closer to mainstream software engineering. Both languages feature a functional language interpreter implementation together with a set of practical tools facilitating systems development (e.g., IDE, debugger, code editor, etc.).

Chapter 3

Agent Platforms

Multiagent system is defined as a set of loosely coupled autonomous software agents which interact together to achieve a common goal. To simplify and speed-up a development process and ensure safe and efficient execution of individual agents software systems called multiagent platforms can be utilized. We can distinguish these platforms according to the level of services they provide to the user varying from full platforms, which are self-contained systems that provide some kind of runtime libraries required for agent applications and application programming interface, to toolkits for creating agent applications which provide engineering support making use of model-driven approaches, graphical editors and other enhancements for designing individual agents. These tools cover graphical UML-like tools or specific agent implementation languages. These models are later transformed into some implementation in a programming language. The resulting applications, unlike in the case of full platforms, usually do not need a dedicated runtime environment or specific libraries. In the following the term multiagent platform will refer to full platforms.

From the point of view of software development multiagent platforms represent a middleware which provides developers and/or administrators with an agent execution environment which takes care about the agent life-cycle management, set of basic services which ensure interoperability, communication, security or service discovery and set of tools which help to monitor status of the system or develop advanced reasoning structures within an agent and interaction schemes among individual agents. From the point of view of interaction among agents we distinguish open and closed multiagent systems. Open systems allow interaction among various types of agents developed by different vendors, which may represent numerous organizations or companies and may act as self-interested. Open systems are e.g. inter-enterprise information systems or supplier-customer trading systems. These systems bring additional requirements on security and interoperability. On the other hand agents forming a closed system usually interact only with a predefined set of other agents which are known to the developer in advance. Example of closed system is an intra-enterprise information system or simulation systems. Simulation systems usually bring more requirements on performance

of underlying multiagent platform, which may in some cases result in optimizations that lead to lower interoperability. In closed systems agents will not try to intentionally harm the rest of the system (unless this feature is studied) and advanced approaches to malicious behavior detection are thus not required.

3.1 FIPA Specifications

Interoperability in open systems is not easy to achieve, as agents can be developed in various programming languages, utilize different agent platforms or executed under several operating systems. In order to create some common specifications that will ensure the interoperability FIPA organization was founded. FIPA specifications represent a collection of standards which are intended to promote the interoperation of heterogeneous agents and services that they can represent. The complete set of specifications can be viewed in terms of different categories: agent communication, agent transport, agent management, abstract architecture and applications. Core category which is most significant for agent interoperability is agent communication. In order to be FIPA compliant, concrete architectural specification must have certain properties. The architecture must include mechanisms for agent registration, agent discovery and inter-agent message transfer. FIPA Abstract Architecture specification defines minimum required elements of agent platform architecture, which are message transport, agent directory, service directory and agent communication language. Agents communicate by exchanging messages which represent speech acts, and which are encoded in an agent communication language. A message is an individual unit of communication between two or more agents. Message includes an indication of the type of communicative act (e.g. inform, query), the agent names of the sender and receiver agents and the content of the message itself. Content of the message can be specified by ontology.

A message transport service supports sending and receiving of messages between agents. Message transport service is responsible for physical delivery of messages to receivers, it manages the socket communication.

The basic role of agent and service directory services is to provide a location where agents/services register their descriptions as directory entries. Other agents/services can search these entries to find partners with which they wish to interact. These services can work as an intra-platform or even expose their records to other platforms to allow interaction among all agents in open systems.

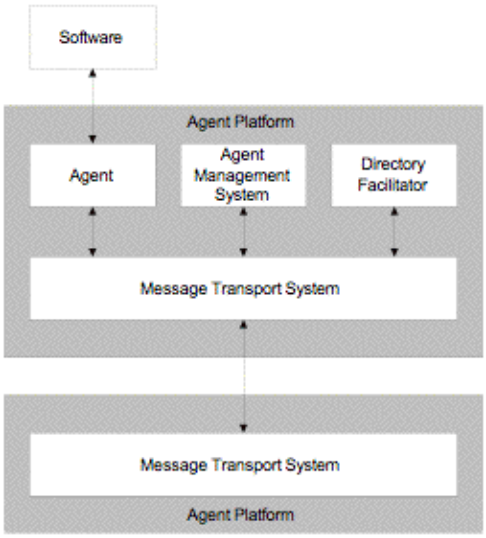

Figure 3.1: Agent Management Reference Model

3.2 FIPA Management Reference Model

FIPA Management Reference Model (as shown in Figure 3.1) provides the normative framework within which FIPA-compliant agents exist and operate. It establishes the logical reference model for creation, registration, location, communication, migration and retirement of agents. Agent in this context is defined as a computational process that implements the autonomous, communicating functionality of an application. An agent must support at least one notion of identity. This notion of identity labels an agent, so that it may be distinguished unambiguously within the agent universe. An agent may be registered at a number of message transport addresses at which it can be contacted.

An Agent Management System (AMS) is a mandatory component of the agent platform. Only one AMS will exist in a single platform. The AMS maintains a directory of unique agent identifiers and transport addresses for agents registered with the agent platform. The AMS offers white pages services to other agents.

An Agent Platform (AP) provides the physical infrastructure in which agents can be deployed. It can be viewed as a kernel responsible for thread, memory and socket management. The AP consists of the machine(s), operating system, agent support software, FIPA agent management components and agents. The internal design of an AP is an issue for agent system developers.

A Directory Facilitator (DF) is an optional component of the AP. It provides yellow pages services to the agents. DF represents an implementation of agent/service directory services.

3.3 Additional Platform Services

Some multiagent platforms support agent mobility. Mobile agents are characterized by code mobility, i.e. mobile robots are not considered as mobile agents. Agent mobility can be used during the simulations to improve the load balancing or in cases when agents operate in environment with unstable conditions (i.e. unreliable communication links, limited computational resources). In open systems agent mobility brings additional requirements on system security. As agents with unknown features and intentions can be deployed on agent platform it is necessary to ensure their thread-safe operation, which will hold the rest of the system harmless against their failure.
Another important feature of multiagent platforms, especially in case of open systems is security. This term represents various aspects of fail-safe operation. From the point of view of agent platform it is necessary to ensure a thread-safe agent execution model, especially in case of mobile agents, as mentioned above. In open systems agents may try to build a model of trust of other agents they interact with to create a set of trustful collaborators. Another important aspect is communication security, which either prevents third-party software from observing message content by its encryption or allows agents to electronically sign messages to prove the origin of the content. Dynamic nature of agent systems brings several problems with distribution of security certificates and private keys.
In total there are at about 150 former and current multiagent platforms and toolkits. Numerous of them are provided under an open license (open source, free license), some are commercial (sale of licenses, authors provide support), rest was developed for use in particular projects and is not available for public use. Most widely used multiagent platforms are Jade [17], JACK [16], AGLOBE [1] or Cougaar [9].

Chapter 4

Agent-based Modeling and Simulation

Agent-based modeling and simulation (ABMS) [24, 4] is a new approach for obtaining insight and foresight regarding the operation of complicated systems. Introduced in mid 1990s and gaining popularity in 2000s, ABMS presents a very different perspective to modeling and simulation. Whereas the traditional approaches (i.e. *dynamical systems*, *system dynamics* and *discrete event-based simulation*) describe a target system top-down in terms of macroscopic, aggregate variables and global relationships and update rules affecting them, ABMS adopts a *microscopic, bottom-up view* and describes target system from the perspective of its constituent units.

The difference can be illustrated e.g. on traffic modeling. Traditional approaches would model the problem in terms of a global graph of queues with add and remove rules; individual cars would be modeled as (homogeneous) items moving between the queues. In contrast, the agent-based approach would model the cars as proactive agents with a locally defined behavior which would depend e.g. on the properties of the road, the position of other cars and characteristics of the driver and other factors. An in-depth discussion of the differences between the different modeling paradigms can be found in [7].

4.1 Simulation Structure

In general, each agent-based simulation consists of two principal components:

- **Agents** – Proactive entities inhabiting the simulation environment (see below). An agent is typically equipped with the ability to (1) *interact* with the environment through sensors and actions, (2) *reason* about information and action (i.e. decide what to do based on what the agent knows and what its goals are), and (3) *communicate* with other agents via sensing messages.
- **Environment** – Agents operate within a defined environment; the topology and state of the environment imposes constraints on what agents can sense (e.g. cannot see through a wall), which actions they can carry out

(e.g. cannot walk through a wall) or with whom an agent can communicate (e.g. cannot send messages when out of signal coverage).

When communication is complex and critical for the simulation, a dedicated model of the communication channel can be created. This is e.g. the case of ad-hoc wireless network used to coordinate team action in a complex environment (e.g. cities).

4.2 Simulation Execution Cycle

A typical execution cycle of an agent-based simulation consists of the following steps:

- **Observe** – Each agent updates its sensory information to reflect the current state of the surrounding environment.
- **Decide** – Considering the agent's internal state, state of the environment as mediated by agent's sensors and messages received from other agents, the agent decides on actions to take.
- **Act** – The agent performs the chosen actions; actions can be internal, external (interacting with the environment and modifying its state) or communication with other agents through sending messages
- **Environment Update** – Changes in the state of the environment caused by other factors than agent activity are applied (e.g. weather, disease spread)

The above simulation cycle is constantly repeated. Depending on the type of the simulation, individual steps can be synchronized between all agents (*synchronous simulation*) or can be carried out independently, potentially with different length for each agents (*asynchronous simulation*).

4.3 Advantages

The bottom-up perspective lends the ABMS a number of important advantages over traditional approaches:

- Captures emergent-phenomena – Emergent phenomena (such as traffic jams, human crowds etc.) result from the interactions of individual entities. Emergent phenomena cannot be reduced to the system's parts – the whole is more than the sum of its parts because of the interactions between the parts – and are therefore beyond the reach of top-down methods.

- Natural, easy-to-maintain models – ABMS is most natural for describing and simulating a system composed of "behavioral" entities – be it a traffic jam, the stock market, voters, or how an organization works. ABMS makes the model seem closer to reality and thus easier to create, maintain and extend.
- Flexibility, scalability and heterogeneity – ABMS can be changed in a variety of useful ways: agents can be easily added, their behavior (reasoning algorithm, rules of interaction etc.) and complexity (degree of rationality, ability to learn etc.) modified. ABMS also make it easy to incorporate heterogeneous agents with varied per-type and per-instance behavior.

The above advantages come at the expense of high computational cost of a typical ABMS. Research on how agent-based simulations can be parallelized and distributed is therefore a very important topic.

4.4 Platforms and Tools

In contrast to the traditional modeling approaches where a number of mature, industry-strength platform exist, the technology for ABMS is still (rapidly) evolving. Two most wide-spread generic ABMS platforms are *NetLogo*[1] and *Repast Simphony*[2]. NetLogo provides an easy-to-use, intuitive design- and runtime framework targeted at non-programmers. Repast, on the other hand, is Java-based and requires good Java programming skills. Recently, ABMS concepts have introduced to some commercial M&S frameworks such as *AnyTime*[3]. A recent survey of ABMS platforms can be found in [26].

[1] `http://ccl.northwestern.edu/netlogo/`
[2] http://repast.sourceforge.net/
[3] `http://www.xjtek.com/`

Chapter 5

Agent Applications in Traffic and Transportation

The multi–agent systems are widely used for planning, control and simulation in various traffic domains. The domain of air traffic management is interesting because of the current and predicted number of flights and way the air traffic is managed nowdays. The air traffic is today controled purely by humans – air traffic controlers and the only criterion of optimality is safety. There are no optimizations of flight trajectory length nor fuel consumption. It is clear that the system of air traffic management needs to be changed using the modern computational techniques because of the predicted increase of flights. To design the system properly, it is necessary to simulate the currently used system, identify the bottlenecks and avoid them in the new system. Within the air traffic management research community, there were developed several approaches and software prototypes used for the simulation of the air traffic management (mostly above US):

- **NASPAC 2.0** (metronaviation + NASA + CSSIINC + FAA) – Developed since 1992, using the queue modeling of the air traffic and event based simulation. The simulation works in several phases using several tools – the tracks are computed first and then other tools are applied on these tracks to compute the data needed for metrics computations
- **FACET** (NASA Ames) – Another system developed by NASA using the queue based modeling techniques. The system studies air traffic in detail – it contains for example studies of air pollution
- **ACES** (NASA Ames + I-a-I + NASA Langley) – Agent–based simulation, developed over 10 years. Contains basic collision avoidance algorithms.

5.1 Simulating Air Traffic Management in AgentFly

AgentFly is a software prototype of multi-agent technology deployment in aerial vehicles air traffic control supporting the free flight concept build on top of the A-globe multi-agent platform. All aerial assets in AgentFly are

modeled as asset containers hosting multiple intelligent software agents. Each container is responsible for its own flight operation. The operation of each vehicle is specified by an unlimited number of time-specific, geographical waypoints. The operation is tentatively planned before take-off without consideration of possible collisions with other flying objects. During the flight performance, the software agents hosted by the asset containers detect possible collisions and engage in peer-to-peer negotiation aimed at sophisticated re-planning in order to avoid the collisions. The aim of this agent deployment is to demonstrate readiness of multi-agent technology for distributed, flexible, and collision-free coordination among heterogeneous, autonomous aerial assets (manned as well as unmanned) with a potential to:

- fly a higher number of aircrafts
- decrease requirements for off-board control operators
- allow a flexible combination of cooperative and non-cooperative collision avoidance

The AgentFly prototype provides:

- distributed model of flight simulation and control
- time-constrained way-point flight planning algorithm avoiding specified no-flight zones,
- flexible collision avoidance architecture, dynamic adjustment to changes in the flight environment
- connectors to external data (Landsat images, airports monitors, no-flight zones, cities),
- 2D/3D visualization including a web-client access component, and
- multiple operator – facilitating real-time control of selected assets

5.2 Collision Avoidance Algorithms in AgentFly

AgentFly provides four distinct collision avoidance (CA) algorithms linked by a flexible mechanism managing the autonomy of individual assets and selecting the best collision avoidance strategy in real time:

- **Rule-based CA algorithm** is a domain dependent algorithm based on the Visual Flight Rules defined by FAA2. Upon the collision threat detection, the collision type is determined on the basis of the angle between the direction vectors of the concerned aircrafts. Each collision

type has a predefined fixed maneuver which is then applied in the re-planning process. Visual flight rule-based changes to flight plans are done by both assets independently because the second asset detects the possible collision with the first asset from its point of view.

- **Utility-based CA algorithm** deploys multi-agent negotiation theories (namely Monotonic Concession Protocol with the Zeuthen Strategy) aimed at finding the optimal CA maneuver. The software agents on each asset generate a set of viable CA maneuvers and compute costs associated with each maneuver (based on e.g. the total length of the flight plan, time deviations for mission way-points, altitude changes, curvature, flight priority, fuel status, possible damage or type of load). The agents negotiate such a combination of maneuvers that minimizes their joint cost associated with avoiding the collision.
- **Multi-party CA algorithm** extends the above presented CA algorithm by allowing several assets to negotiate about collective CA avoidance maneuver. This algorithm minimizes the effects of CA maneuvers causing conflicts in future trajectories with other flying assets. While requiring more computational resources, this strategy has shown to provide more efficient free-flight collision free trajectories.
- **Non-cooperative CA algorithm** supports collision avoidance in the case when communication between aircrafts is not possible. Such a situation can arise e.g. when on-board communication devices are temporarily unavailable or when an asset avoids a hostile flying object. This algorithm is based on modeling/predicting the future airspace occupancy of the non-cooperative object and representing it in terms of dynamic no-flight zones. Based on this information, the algorithm performs continuous re-planning.

Chapter 6

Agent Applications in Network Security Monitoring

The session will introduce the concept of security monitoring in general and define the problems of system monitoring and autonomous analysis in an uncertain, adversarial environment. Then, using this general theoretical framework, we will use the example of network monitoring and intrusion detection to illustrate the theoretical concepts on a real-world example.

6.1 Motivation

The problem of security monitoring is well defined in many areas of both physical [2] and electronic security [28]. Monitoring is essential and indispensable component of any security infrastructure, as it allows to manage the system, discover possible failures of prevnetive/perimeter devices and address them before the attack causes major harm or disruption. On the other hand, security monitoring is an attractive target for any sophisticated attacker, as its compromise is necessary for successful execution of attacks against highly secure, multi-level defences. With the increasing role of distributed and autonomous computational elements in the security infrastructure (ranging from intelligent motion sensors and shape identifying cameras to sophisticated intrusion detection systems in computer security), a new attack frontier emerges against the intelligent algorithms embedded in the monitoring infrastructure.
The use of techniques from the field of agent-based computing [43], game theory [27] and autonomic computing [37] can make the algorithms more robust w.r.t. intentional manipulation of their models [10, 3]. Our talk will cover the basic attack techniques and outline possible defence strategies against them, before presenting more specific deployment scenarios from the network security field.

6.2 Attacks

The attacks can be broadly categorized into several classes, in increasing level of sophistication:

- Passive attacks on sensors - prevent the sensor from noticing the object. Typical examples are insertion/evasion [30] in network security, or plastic foils that prevent the heat from the attacker from reaching the IR movement sensor.

- Active attacks on sensors - actively modify sensor or environmental characteristics to render it ineffective. Examples include all kinds of electronic attacks, such as jamming [2].

- Passive attacks on pattern matching/feature extraction - attack the ability of the detection system to extract meaningful information from the data captured by the sensor. Examples include polymorphic malware [28] in network security, shape disrupting camouflage in military applications and many others.

- Active attacks on pattern matching/feature extraction: These attacks try to use the knowledge of the feature extraction algorithm to actively modify its characteristics. Examples are concurrent attacks in computer security, where a large attack hides the existence of a more dangerous small attack, or the attacks involving traffic shaping.

- Passive attacks on detection model - are designed to avoid attacker's detection by inferring the internal state of the detection algorithm and then adapting the attack behavior so that it is still effective as an attack, but difficult to detect by the algorithm. The examples include learning, adaptive malware, that can adapt to detection algorithm capability [20].

- Active attacks on detection algorithm model - are designed to modify the internal state of the detection algorithm, so that the attacker can then exploit this state. A nice example of this technique is adversarial traffic shaping in network security [40].

- Denial of service/capacity attacks - are designed to overwhelm the operator using the system with an unmanageable number of attacks. Again, these techniques are frequently used in electronic attacks [30], as well as in the physical attacks on alarm systems [2].

6.3 Defence Strategies

Available defence strategies to the above cases are typically inspired by joining the techniques from the field of game theory, autonomic systems and multi-agent systems in order to deliver a system that is both difficult to

model and understand for the opponent and still delivers consistent detection results in a wide range of scenarios. Adaptivity in such cases needs to be driven strategically, take into account possible manipulation by the opponent and play rich randomized strategies that are hard to predict and exploit.

Chapter 7

Agent Applications in Production Planning

An agent based production planning and scheduling system is a new and very efficient approach, based on methods of artificial intelligence. A system is implemented as a community of agents that communicate and cooperate one with each other and together create a plan.
Contrary to a standard centralized planning system, multi agent system supports an easy integration of an existing software and hardware. Each existing system that should be involved in the planning process is "agentified" - equipped with an interface that allows them to behave as an agent. The multi agent technology is scalable to a wide area of planning problems from small factories to large enterprises with hundreds of connected units. Another advantage is an acceleration of the planning process that is caused by parallel computations, by employment of the methods of the distributed artificial intelligence and by an effective re-planning. When some part of the plan cannot be executed, only a minimal part of the plan have to be changed and only necessary agents are involved in the re-planning and re-configuration.
The multi-agent system is not suitable for planning and scheduling only. When it is extended by the production unit models and real-time feedback it is able to provide also efficient simulation. When agentification of the real equipment is made, the same interface is used for connecting simulation model to emulate the behaviour of the equipment for various tests end experiments. Agent based simulation provides the estimation of performance and workflow progression for various planning/scheduling strategies. Agent based modelling and simulation is suitable mainly for dynamic self-organizing production processes. The simulation based on defined roles and data base refined using real-time feedback provides valuable prediction of the production system response to various control action, environment changes or faults.

7.1 Agent-based Planning for Industry

Agent technologies are well suited in the domain of simulation and planning of the manufacturing processes. Agent technologies provide high flexibility and modular architecture. The Multi-agent system is suitable for i) middle and high level planning of the manufacturing processes on short or long-term

period with possible incorporating external suppliers, ii) low-level real-time control and planning with production feedback, iii) configuration and simulation of the manufacturing process to find the most expedient structure and plan. The concept of agentification allows operating with existing software/hardware solutions as well as human resources and robots. The concept of agent may be applied on different levels of the production process [23]:

- On the lowest level, agents maintain inseparable functional units within one shop-floor (in case of enterprise, the real-time holonic control is linked with the physical devices;
- On the level of one plant structure member intra organization management (shop-floor inside an enterprise - also called intra-enterprise level) the agent-based planning and scheduling applies;
- On the level of the inter-enterprise cooperation (also called extra-enterprise level) agents provide integration into collaborative networks.

Any complex physical system consists of a number of partially independent components that are of various types and that are working parallel. Therefore, the simulation and modeling of such multi-component systems (e.g. workflow) can naturally utilize distributed technologies like multi-agent systems [8].

7.2 Industrial Agent-based Applications

The potential and discussion of the MAS implementations in the manufacturing domain can be found in [31]. Examples of utilization of Multi-agent technology in domain of Production Planning and Control (PPC) are systems ExPlanTech and ExtraPlanT. The ExPlanTech system demonstrates successful deployment of multi-agent technology in the domain of shop-floor production planning in automotive industry [32]. The suppliers and external partners integration has been extended in ExtraPlanT system [14]. In these systems, the agents represent the production workshops and cooperate to prepare the production plan on the intra-enterprise (ExPlanTech) and extra-enterprise (ExtraPlanT) levels. The real online production feedback helps to adapt the plan to keep the plan constrains consistent and react fast to the unexpected events. The example of agent-based control system is MAST [22] that provides the support for integrating RFID products identification [46]. In the MAST, the agents represent machines and products and negotiate to dynamically optimize whole production process.

Figure 7.1: Examples of industrial agent-based applications.

Another application area of MAS is modelling and simulation of production processes. The SimPlanT [45] is the agent-based system that utilizes autonomous models of production entities and event-based simulation for analyzing production flows and quality of plan variants. The Hyres system [39] uses agent-based models to diagnose the complex production process (based on alarms observed in the system), analyses the potential impact of the observed problems, and provides the root cause analyses. The agent-based Decision Support System for Virtual Organization simulations can be found in [15]. The examples of the industrial agent-based applications are depicted on Figure 7.1

Bibliography

[1] AGLOBE. *AGlobe Agent Platform.* `http://agents.felk.cvut.cz/aglobe`, 2010.

[2] Anderson, R. J. *Security Engineering: A Guide to Building Dependable Distributed Systems.* Wiley, January 2001.

[3] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., Tygar, J. D. Can machine learning be secure?. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, s. 16–25, New York, NY, USA, 2006. ACM.

[4] Bonabeau, E. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, sv. 99, č. 3, s. 7280–7287, Květen 2002.

[5] Bordini, R. H., Hübner, J. F., Wooldridge, M. *Programming Multi-agent Systems in AgentSpeak Using Jason.* Wiley Series in Agent Technology. Wiley-Blackwell, 2007.

[6] Bordini, R. H., Dastani, M., Dix, J., Seghrouchni, A. E. F. *Multi-Agent Programming Languages, Platforms and Applications.*, vol. 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* Kluwer Academic Publishers, 2005.

[7] Borshchev, A., Filippov, A. From system dynamics and discrete event to practical agent based modeling: Reasons, techniques, tools. In *Proceedings of the 22nd International Conference of the System Dynamics Society*, s. 25–29, 2004.

[8] Bölöni, L., Boloni, L., Marinescu, D. C., Rice, J. R., Tsompanopoulou, P., Vavalis, E. A. *Agent Based Networks for Scientific Simulation and Modeling.*, 1999.

[9] Cougaar. *Cognitive Agent Architecture.* `http://www.cougaar.org`, 2010.

[10] Dalvi, N. N., Domingos, P., Mausam, Sanghai, S. K., Verma, D. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, s. 99–108. ACM, 2004.

[11] Dastani, M., van Riemsdijk, M. B., Meyer, J.-J. *Programming Multi-Agent Systems in 3APL.*, chapter 2, s. 39–68 Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [6], 2005.

[12] Fisher, M., Hepple, A. *Executing Logical Agent Specifications.*, chapter 1, s. 3–27 In Bordini et al. [25], 2009.

[13] Hindriks, K. V. *Programming Rational Agents in GOAL.*, chapter 4, s. 119–157 In Bordini et al. [25], 2009.

[14] Hodík, J., Bečvář, P., Pěchouček, M., Vokřínek, J., Pospíšil, J. Explantech and extraplant: multi-agent technology for production planning, simulation and extra-enterprise collaboration. *International Journal of Computer Systems Science and Engineering*, sv. 20, č. 5, s. 357–367, 2005.

[15] Hodík, J., Vokřínek, J., Hoffman, R. Decision support system for virtual organization management. In *Innovative Production Machines and Systems, Procedings of the Third I*PROMS Virtual International Conference, 2-13 July, 2007*, s. 85–90, Dunbeath, 2008. Whittles Publishing.

[16] JACK. *JACK Agent Platform.* `http://www.agent-software.com`, 2010.

[17] JADE. *Java Agent Development Framework.* `http://jade.tilab.com`, 2010.

[18] Jones, R. M., Wray III, R. E. Comparative analysis of frameworks for knowledge-intensive intelligent agents. *AI Magazine*, sv. 27, č. 2, s. 45–56, 2006.

[19] Kaelbling, L. P., Rosenschein, S. J. *Designing Autonomous Agents.*, chapter Action and planning in embedded agents, s. 35–48 The MIT Press, Cambridge, MA., 1990.

[20] Kayacik, H. G., Zincir-Heywood, A. N. Mimicry attacks demystified: What can attackers do to evade detection?. *Privacy, Security and Trust, Annual Conference on*, sv. 0, s. 213–223, 2008.

[21] Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R. B. Golog: A logic programming language for dynamic domains. *J. Log. Program.*, sv. 31, č. 1-3, s. 59–83, 1997.

[22] Marík, V., Vrba, P., Fletcher, M. Agent-based simulation: Mast case study. In *BASYS*, s. 61–72, 2004.

[23] Marík, V., Lazanský, J. Industrial applications of agent technologies. *Control Engineering Practice*, sv. 15, č. 11, s. 1364 – 1380, 2007 Special Issue on Manufacturing Plant Control: Challenges and Issues - INCOM 2004, 11th IFAC INCOM'04 Symposium on Information Control Problems in Manufacturing.

[24] Miller, J. H., Page, S. E. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life (Princeton Studies in Complexity).* Princeton University Press, Březen 2007.

[25] Bordini, R. H., Dastani, M., Dix, J., Fallah-Seghrouchni, A. E., editors *Multi-Agent Programming: Languages, Tools and Applications.* Springer, Berlin, 2009.

[26] Nikolai, C., Madey, G. Tools of the Trade: A Survey of Various Agent Based Modeling Platforms. *Journal of Artificial Societies and Social Simulation*, sv. 12, č. 2, s. 2, 2009.

[27] Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. V. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

[28] Northcutt, S., Novak, J. *Network Intrusion Detection: An Analyst's Handbook.* New Riders Publishing, Thousand Oaks, CA, USA, 2002.

[29] Pokahr, A., Braubach, L., Lamersdorf, W. *Jadex: A BDI Reasoning Engine.*, chapter 6, s. 149–174 Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [6], 2005.

[30] Ptacek, T. H., Newsham, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[31] Pěchouček, M., Rehák, M., Maříк, V. Expectations and deployment of agent technology in manufacturing and defence: case studies. In Pěchouček, M., Steiner, D., Thompson, S. G., editors, *AAMAS Industrial Applications*, s. 100–106. ACM, 2005.

[32] Pěchouček, M., Vokřínek, J., Bečvář, P. Explantech: Multiagent support for manufacturing decision making. *IEEE Intelligent Systems*, sv. 20, č. 1, s. 67–74, 2005.

[33] Rao, A. S., Georgeff, M. P. BDI-agents: from theory to practice. In *Proceedings of the First Int. Conference on Multiagent Systems*, San Francisco, 1995.

[34] Rao, A. S., Georgeff, M. P. Modeling Rational Agents within a BDI-Architecture.. In *KR*, s. 473–484, 1991.

[35] Rao, A. S., Georgeff, M. P. An Abstract Architecture for Rational Agents.. In *KR*, s. 439–449, 1992.

[36] Rao, A. S. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In de Velde, W. V., Perram, J. W., editors, *MAAMAW*, vol. 1038 of *Lecture Notes in Computer Science*, s. 42–55. Springer, 1996.

[37] Rehák, M., Staab, E., Fusenig, V., Pechoucek, M., Grill, M., Stiborek, J., Bartos, K., Engel, T. Runtime monitoring and dynamic reconfiguration for intrusion detection systems. In Kirda, E., Jha, S., Balzarotti, D., editors, *Recent Advances in Intrusion Detection, 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, 2009. Proceedings*, s. 61–80, 2009.

[38] Rodney, A., Brooks, R. A. How to build complete creatures rather than isolated cognitive architectures. In Vanlehn, K., editor, *Architectures for Intelligence*, s. 225–240. Lawrence Erlbaum Associates, 1991.

[39] Rollo, M., Novák, P., Kubalík, J., Pěchouček, M. Alarm root cause detection system. In Camarinha-Matos, L. M., editor, *Emerging Solutions for Future Manufacturing Systems*, s. 109–116. New York: Springer, 2004.

[40] Rubinstein, B. I. P., Nelson, B., Huang, L., Joseph, A. D., hon Lau, S., Taft, N., Tygar, J. D. Evading anomaly detection through variance injection attacks on pca. In Lippmann, R., Kirda, E., Trachtenberg, A., editors, *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*, vol. 5230 of *Lecture Notes in Computer Science*, s. 394–395. Springer, 2008.

[41] Russell, S. J., Norvig, P. *Artificial Intelligence: A Modern Approach (2nd Edition).* Prentice Hall, December 2002.

[42] Shoham, Y. Agent-oriented programming.. *Artif. Intell.*, sv. 60, č. 1, s. 51–92, 1993.

[43] Shoham, Y., Powers, R., Grenager, T. If multi-agent learning is the answer, what is the question?. *Artif. Intell.*, sv. 171, č. 7, s. 365–377, 2007.

[44] Thomas, S. R. *PLACA, an agent oriented programming language.* PhD thesis, Stanford, CA, USA, 1993.

[45] Vokřínek, J., Pavlíček, D., Šmerák, R. Simulation of manufacturing processes using multi-agent technology. In Pham, D., Eldukhri, E., Soroka, A., editors, *Intelligent Production Machines and Systems*, s. 461–466. Elsevier Science, 2005.

[46] Vrba, P., Macurek, F., Marík, V. Using radio frequency identification in agent-based manufacturing control systems. In *HoloMAS*, s. 176–187, 2005.

[47] Winikoff, M. *JACK^{TM} Intelligent Agents: An Industrial Strength Platform.*, chapter 7, s. 175–193 Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations* [6], 2005.

[48] Wooldridge, M. *Reasoning about rational agents.* MIT Press, London, 2000.

Centrum pro rozvoj výzkumu pokročilých řídicích a senzorických technologií
CZ.1.07/2.3.00/09.0031

Ústav automatizace a měřicí techniky
VUT v Brně
Kolejní 2906/4
612 00 Brno
Česká Republika

http://www.crr.vutbr.cz
info@crr.vutbtr.cz