DOI:10.2298/CSIS110214013B

Software Agents: Languages, Tools, Platforms

Costin Bădică[1], Zoran Budimac[2], Hans-Dieter Burkhard[3], and Mirjana Ivanović[2]

[1]Software Engineering Department, Faculty of Automatics, Computers and Electronics,
Bvd.Decebal, Nr.107, Craiova, RO-200440, Romania
badica_costin@software.ucv.ro
[2] Faculty of Sciences, Department of Mathematics and Informatics
Trg Dositeja Obradovica 4, 21000 Novi Sad, Serbia
{zjb, mira}@dmi.uns.ac.rs
[3]Humboldt University, Institute of Informatics,
Rudower Chaussee 25, D-12489 Berlin, Germany
hdb@informatik.hu-berlin.de

Abstract: The main goal of this paper is to provide an overview of the rapidly developing area of software agents serving as a reference point to a large body of literature and to present the key concepts of software agent technology, especially agent languages, tools and platforms. Special attention is paid to significant languages designed and developed in order to support implementation of agent-based systems and their applications in different domains. Afterwards, a number of useful and practically used tools and platforms available are presented, as well as support activities or phases of the process of agent-oriented software development.

Keywords: agent technologies, agent programming languages, agent platforms.

1. Introduction

The metaphor of “intelligent software agents” as basic building blocks for the development of new generation intelligent software systems triggered both theoretical and experimental computer science research aiming to develop new programming languages for agent systems. Fifteen years ago [64] software agent technology has been recognized as a rapidly developing area of research and one of the fastest growing areas of information technology.

In our opinion, the main achievement of this trend of research was the development of new programming models that address both the basic features of agenthood (autonomy, reactivity, proactivity and social abilities) as well as more advanced, human-like features usually collectively coined in the agent literature as “mental attitudes” (beliefs, desires, intentions, commitments), following the model of “intentional systems” introduced by the philosopher Daniel Dennett in 1971 to explain behavior of rational agents.

Agent oriented technologies, engineering of agent systems, agent languages, development tools and methodologies are an active and emergent research area and agent development is getting more and more interesting. There are many approaches, theories, languages, toolkits, and platforms of different quality and maturity which could be applied in different domains.

Our motivation and the main goal of the paper are to bring a survey in the field of agent technology and to cover different aspects of agents. Agents, agent-oriented programming (AOP), and multi-agent systems (MAS) introduce new and unconventional concepts and ideas. Still, there is a number of definitions of the term 'agent' that include a property common to all agents: *agent acts on behalf of its user,* as well as a lot of additional properties: agent communicates with other agents in a multi-agent system; acts autonomously; is intelligent; learns from experience; acts proactively as well as reactively; is modeled and/or programmed using human-like features (beliefs, intentions, goals, actions, etc.); is mobile, and so on.

After more than two decades of scientific work in the field, the challenge is to include agents in real software environments and widely use the agent paradigm in mainstream programming. One way to facilitate this is to provide agent-oriented programming languages, tools and platforms.

Pioneering work is done by the *Foundation for Intelligent Physical Agents (FIPA).* "FIPA was originally formed as a Swiss based organization in 1996. Since its foundations, FIPA has played a crucial role in the development of agents standards and has promoted a number of initiatives and events that contributed to the development and uptake of agent technology. Furthermore, many of the ideas originated and developed in FIPA are now coming into sharp focus in new generations of Web/Internet technology and related specifications." (cf. [116]). Since 2005, FIPA is the standards organization for agents and multi-agent systems of the IEEE Computer Society standards organization.

Our recent overview of the agent programming literature revealed a number of trends in the development of agent programming languages. These trends follow the main achievements of computer science disciplines that are traditionally directly connected to multi-agent systems, i.e. formal methods, object-oriented programming, concurrent programming, distributed systems, discrete simulation, and artificial intelligence. Adding on top of that the metaphor of "humanized agents" with roots in psychology research, by regarding them as intentional systems that are endowed with mental states, we can get a panoramic view of the current status of the world of agent programming languages, tools and platforms. The whole paper or some sections of it could be extremely useful and give more insights into the domain for a wide range of readers. PhD students and young researchers can find plenty of useful information and state-of-the-art in the domain of available languages and platforms for programming software agents. Professionals in different companies who are willing to apply this new, promising technology in everyday programming and implementation of real world applications based on agent technology, could find the paper very helpful. Undergraduate students who like to widen their traditional knowledge and be introduced to

modern trends in programming can use it as an additional reading material. Moreover, all of them can find a valuable source of references and suggestions for further reading.

The rest of the paper is organized as follows. The second section attempts to give an overview of all essential notions, issues and concepts related to agents and agent technology which are used in other chapters of the paper. Thereinafter, it makes the distinction between single agent and multi-agent systems, goes through the broad spectrum of agent properties, discusses the most acknowledged classifications of software agents, presents the most well-known agent architectures, and explores the two most important agent communication approaches. Section three lists and discusses standard languages and several prototype languages that have been proposed for constructing and implementing agent-based systems. Afterwards, section four presents a number of tools and platforms that are available to support activities or phases of the process of agent-oriented software development. The last chapter gives some concluding remarks.

2. What is a software agent?

2.1. Introduction

Over the last years, many researchers in different fields have proposed a large variety of definitions for the term “agent”. The common understanding is that it is an entity which “acts autonomously on behalf of others”. Even if we restrict ourselves to computer science, there are a lot of different definitions and a lot of different fields where agents are used. It started 30 or more years ago in (Distributed) Artificial Intelligence. With the arrival of the Internet and with the dissemination of computer games, the notion of agents has become broadly used even by non-experts, e.g. for electronic marketing, assistance systems, search engines, chatter bots etc, or as constituents of larger software projects. For the latter ones, it is useful to distribute the overall tasks to autonomous entities and to organize a framework of cooperation and interaction in a multi-agent system. Agents are typical inhabitants of open systems like the Internet. Open systems have been characterized by Hewitt [58] already in the 80’s as systems with continuous availability, extensibility, modularity, arm-length relationships, concurrency, asynchronous work, decentralized control, and inconsistent information.

Michael Coen [126] puts very small restrictions on a program to be considered as an agent: "... programs that engage in dialogs and negotiate and coordinate transfer of information." In the IBM [127], intelligent agents are defined as: "...software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires.". The Software Agents Group at MIT [128] compares

software agents to conventional software and emphasizes the following differences: "Software agents differ from conventional software in that they are long-lived, semi-autonomous, proactive, and adaptive.". More detailed is the so-called “weak notion of agency” by Wooldridge and Jennings [104], [105]. They define an agent as "... a hardware or (more usually) software based computer system that enjoys the properties:

- **autonomy**: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- **social ability**: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- **reactivity**: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- **proactiveness**: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

A definition in the sense of “strong notion of agency” is [105]: "An agent is a computer system that, in addition to having the properties identified in the definition of weak agent, is either conceptualized or implemented using concepts that are more usually applied to humans (knowledge, obligations, beliefs, desires, intentions, emotions, human-like visual representation, etc.)."

The “strong notion of agency” corresponds to the usage in the field of artificial intelligence (AI). These systems are often specified using human mental categories: beliefs, plans, goals, intentions, desires, commitments, etc. Shoham [90] claims that the use of mental categories in agent specification is justified only under the following conditions:

- mental categories are precisely defined using some formal theory,
- agent has to obey that theory,
- every mental category used in an agent specification has to give some benefit.

A collection of various agent definitions, based on the weak notion of agency, can be found in [52]. It is not the aim of this paper to give a unique definition of an agent. Instead, the reader will find that the different tools presented in the rest of the paper correspond to different notions. Some more relevant concepts are introduced in the following sections.

2.2. Agent Classification and Architectures

General classifications in the agent community [105] distinguish between reactive architectures and deliberative architectures. Reactive architectures are considered as simple controls, while deliberative architectures implement complex behavior including mental attitudes (goals etc.) and planning, based on symbolic representations and models. Hybrid architectures are combinations of both reactive control on the “lower level” for fast responses and deliberative control on the “higher level” for long-term planning.

A technologically better classification addresses the possible "states" of an agent, similar to the approach given in [86]. A state is a snapshot of the system (e.g. the content of the memory) at a certain time point on a discrete time scale. Transitions describe the state changes between two time points. For agents, the time scale is usually chosen according to the sense-think-act-cycle. This cycle consists of

- processing incoming information ("sense", e.g. parsing messages from other agents, analyzing human requests, possibly in natural language etc.)
- more or less complex decision procedures ("think", e.g. by simple decision or rules, or by deliberation, planning etc.)
- sending outgoing information ("act", sending messages to other agents, preparing answers in a human-like style etc.).

An internet agent may in one cycle get a customer request, update its database and send an answer. The state is the content of the database at the end of this cycle. An agent may have different cycles at different time scales: A search engine may answer search requests more frequently than its index machine gets updated. "This creates a need for synchronization efforts which can be facilitated e.g. by different layers."

A behavior (architecture) is called *stimulus-response* behavior if there are no different states at all. The response of the agent to an input is always the same (if there is some probabilistic component, then the distribution is the same). It can be produced e.g. by a fixed set of rules, by an input-output table or by a neural network. It doesn't matter if the response routine is a simple or a complex one. A search engine may perform extensive search over large databases and a lot of effort to rank the results. If nothing is stored after answering, then the same calculations yielding an identical answer will be performed every time for the same request.

If the responses of an agent to identical requests are different, then they depend on the state of the agent. The search engine may maintain profiles of its users such that the answers depend on the stored profiles. The profile is updated every time the user makes a request, i.e. the state of the agent is changed. It is useful to distinguish between different kinds of states according to their contents:

The so-called *belief* or *world model* stores internal representations about the situation in the environment for later usage. It is updated according to the sensor inputs. It is called belief because it needs not to be true knowledge (e.g. the profile of a user calculated only by the available user inputs needs not represent her or his true preferences).

Future directed states are created by "mental attitudes" related with decision processes. They are named as *goals*, *plans* etc., and they guide the future deliberation and actions (towards a formerly chosen goal). A trading agent may have the goal of an optimal transaction. For that, it may develop a plan for searching appropriate offers from databases and for negotiation with other agents.

As introduced above, agents are called "*stimulus-response*" if they do not have states. Nevertheless, their decision procedures might be very complex, e.g. in complex information systems. Agents with states may have a world

model (belief) and/or future-directed state components like goals and plans. Their deliberation process considers updates of the world model and commitment procedures for selecting goals and constructing plans. Actually, it is up to the programmer to decide if mental notions are used for data constructs. Sometimes, very simple agents come with mental attitudes. There is nothing wrong with it if it helps for better understanding.

The popular *BDI architecture* is inspired by the work of the philosopher [16]. BDI stands for *belief*, *desire* and *intention*. Belief describes the world model as above, while desires and intentions are future-directed mental notions. Bratman argues that the mental notion of goals is not sufficient to express complex future-directed behavior. A rational agent should adopt only goals which it believes to be achievable, i.e. not in conflict with the belief and not in conflict with each other. However, before committing to a goal, the agent may have different desires, which may be in conflict. A human may at the same time have conflicting desires, e.g. go to home, to be at the beach, to ride a bicycle, and to drive a car. Then he has to make a commitment, to choose which desires to adopt as intentions. Rational behavior demands to select only non conflicting options, e.g. to go by bicycle to the beach.

In BDI architectures, desires are used as preliminary stages of possible intentions: first the agent collects desirable options, and then it selects some of them as intentions e.g. through ranking, while avoiding conflicts between intentions. Then it performs appropriate actions to achieve the intentions.

Not all the so-called BDI-architectures really implement Bratman's ideas. In some cases, the agent simply selects a single desire and calculates an appropriate action sequence (called intention) which fulfills this desire. In such a case, a desire is in fact a goal, and the intention is the related plan to achieve the goal.

A lot of theoretical work using multimodal temporal logics has been performed for the foundation of deliberative agent controls and some of them resulted in executable formalisms like MetateM ([48]).

2.3. Robots and Software Agents

Robots are often considered as hardware controlled by a software agent acting as the brain. The sensors and actuators of the robot provide the input and output for the agent. This works well in simple settings, but it poses problems for more complex robots in real environments. Control of such robots is more than information processing: parts of such robots coordinate not only by messages, but by physical interactions too. It is very difficult or even impossible to model the physical dependencies in terms of information processing. However, those relations can be used directly by clever design. Modern robotic approaches are inspired by biology and use local sensor-actor loops, etc. "The key observation is that the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough." [109] This paradigm is known as *behavioral robotics*, *biologically inspired robotics* etc.

Related robot controls are able to perform surprisingly complex tasks. Their behavior *emerges* from the physical *situatedness* of an *embodied* entity. An approach that exploits situated automata is described in [66]. Pattie Maes [72] has developed an agent architecture that is composed of modules organized into a network. The *Subsumption Architecture*, based on behaviors, is the best-known architecture of this kind. Brooks built many robots (based on four principles) using an AI approach [25], [26]:

- *Situatedness* - The robots are situated in the world.
- *Embodiment* - The robots have bodies and experience the world directly - their actions are part of a dynamics with the world and have immediate feedback on their own sensations.
- *Intelligence* - They are observed to be intelligent - but the source of intelligence includes: computational engine, situation in the world, the signal transformations within the sensors, and the physical coupling of the robot with the world.
- *Emergence* - The intelligence of the system emerges from the system's interactions with the world and from sometimes indirect interactions between its components.

A behavior is implemented as a simple state machine with few states (not representations of the outside world). Behaviors can overwrite each other (subsumption). Brooks' robots built from connected behaviors are capable of performing some complex tasks with relatively simple programming [24]. Maes has shown that the same ideas can also be exploited in the design of software agents [73]. The behavioral agent architectures are sometimes considered as prototypes of reactive architectures [105]). While single behaviors are simple, their hierarchical combination into a more complex behavior becomes more and more complicated. Because of that, the *hybrid* architectures combine low-level approaches with classical reasoning approaches in hierarchical architectures. Such architectures may consist of several levels, where low levels can use behavioral (and possibly subsymbolic) architectures, and higher levels are usually deliberative (symbolic) ones. Symbolic modeling becomes necessary when sensing does not give enough information, or when planning is really needed.

Because of the “physics in the loop”, the assumptions usually connected with software agents are not fulfilled: Physical components do not behave like objects (or agents). This difference is recently stressed by the investigation of so-called Cyber Physical Systems [110], which are considered as distributed systems where the components perform information processes as well as physical processes. The interaction between the components is of physical and computational nature, as well.

2.4. More Features of Agents

Depending on different usages of agents, they can have a lot of different features. Such features are often used for classification as well. We have already discussed different basic architectures. Next we describe mobility,

size intelligence and ability to adapt and to learn. Relations to other agents are described later in the section on Multi-Agent Systems. A collection of agent features will be given in table 1.

Mobility - Agents can be *static* or *mobile*. Static agents are permanently located at one place, while mobile agents can change their location. When a static agent wants some action to be executed at a remote site, it will send a message to an agent at that location with the request for the action. In a similar situation, a mobile agent would transmit itself to the remote site and invoke the action execution. There are a lot of benefits from usage of mobile agents [27] but if we wanted to get all of these benefits without a mobile agent, we would need a large amount of work and it would be practically almost impossible [1]. The advocated utility of mobile agents is to support optimization of sophisticated operations that may require strong interactivity between components or special computing facilities as encountered e.g. in negotiation, network management and monitoring, and load balancing for scientific computing. Mobility of software agents is closely related to the problem of code mobility in distributed programming with applications in operating systems and computer networks. Some problems related with mobile agents concern security and safety. A good overview of code mobility paradigms can be found in the reference paper [53].

Size and Intelligence - Agents can be of various sizes and can possess various amounts of intelligence. Generally, intelligence of a software agent is proportional to its size, so we can distinguish: *big-sized*, *middle-sized* and *micro agents*. It is difficult to make clear boundaries among these categories.

1. A big-sized agent occupies and controls one or more computers. It possesses enough competence to be useful even if it acts alone, without the other agents in a MAS. A big-sized agent can be as big and as intelligent as an expert system [63] with competences for expert problem solving, e.g. distributed medical care or plane ticket reservation.
2. A middle-sized agent is the one that is not useful without the other agents in a MAS or without additional software [6], [7], [8]. However, it is able to perform some non-trivial task(s). A user-interface agent that acts without other agents and performs some simple actions can also be classified as a middle-sized agent. Mobile agents are usually middle-sized agents.
3. Micro agents (also called the Society of Mind agents) [77] do not possess any intelligence. Minsky followed the idea that the intelligence emerges as a global effect of the overall activity of many simple and unintelligent agents.

Adaptation – Adaptive agents can adapt their behavior to different situations and changes in the environment. For example, a navigation system can adapt to changes in traffic (e.g. a traffic jam) and propose alternative routes. This makes adaptive agents more robust to non-predicted changes in a dynamic environment.

Learning - Agents can use learning capabilities for better performance. Learning can be done online, e.g. by data mining from data which are constantly collected through interaction with users (e.g. for profiles). Offline

learning refers to training processes (e.g. for pattern recognition) prior to productive agent usage.

Agents may possess many features in various combinations. The following table is a slightly modified collection from [54]:

Table 1. Agent's features

Adaptivity	*Agents can adapt to unpredicted changes.*
Autonomy	An agent can act without direct intervention by humans or other agents and that it has control over its own actions and internal state
Benevolence	It is the assumption that agents do not have conflicting goals and that every agent will therefore always try to do what is asked of it.
Character (personality)	An agent has a well-defined, believable "personality" and emotional state.
Competitiveness	An agent is able to coordinate with other agents except that the success of one agent may imply the failure of others.
Cooperation or collaboration	An agent is able to coordinate with other agents to achieve a common purpose; non-antagonistic agents that succeed or fail together.
Coordination	An agent is able to perform some activity in a shared environment with other agents. Activities are often coordinated via plans, workflows, or some other process management mechanism.
Credibility	An agent has a believable personality and emotional state.
Deliberation	A deliberative agent decides for its actions by reasoning processes which may involve mental categories like goals, plans etc.
Embodiment	An embodied agent can interact with its environment by physical processes. This allows for emergent controls guided by sensor data without internal representations.
Emergent behavior	More complex behavior emerges by interaction of (simple) agents with each other (swarm intelligence) or with the environment (embodied agents, situated agents).
Flexibility	The system is responsive (the agents should perceive their environment and respond in a timely fashion to changes that occur in it), pro-active and social.
Goal directed	Agent behavior is guided by mental qualities like goals, which are results of deliberation. Then the agent tries to achieve the goal by appropriate actions.

Hybrid architecture	Combination of different architectures. Often with simple (reactive, stimulus response) control for low level behavior and deliberative control for high level behavior.
Inferential capability	An agent can act on abstract task specification using prior knowledge of general goals and preferred methods to achieve flexibility; goes beyond the information given, and may have explicit models of self, user, situation, and/or other agents.
Intelligence	An agent's state is formalized by knowledge and the agent interacts with other agents using symbolic language.
Interpretation ability	An agent is interpretive if it can correctly interpret its sensor readings.
"Knowledge-level" communication ability	The ability to communicate with persons and other agents with language more resembling human-like "speech acts" than typical symbol-level program-to-program protocols.
Learning	An agent is capable of learning from its own experience, its environment, and interactions with others.
Mobility	an agent is able to transport itself from one machine to another and across different system architectures and platforms.
Prediction ability	An agent is predictive if its model of how the world works is sufficiently accurate to allow it to correctly predict how it can achieve the task.
Proxy ability	An agent can act on behalf of someone or something acting in the interest of, as a representative of, or for the benefit of, some entity.
Personality (character)	An agent has a well-defined, believable "personality" and emotional state.
Proactiveness	An agent does not simply act in response to its environment; it is able to exhibit goal-directed behavior by taking the initiative.
Rationality	It is the assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit.
Reactivity	An agent receives some form of (sensory) input from its environment, and it performs some action that changes its environment in some way.
Resource limitation	An agent can only act as long as it has resources at its disposal. These resources are changed by its acting and possibly also by delegating.

Reusability	Processes or subsequent instances can require keeping instances of the class 'agent' for an information handover or to check and to analyze them according to their results.
Ruggedization	An agent is able to deal with errors and incomplete data robustly.
Sensor-actor coupling	Agents act by (direct) connections between sensors and actors. This can be used for reactive controls.
Situatedness	An agent (robot) is situated in its environment. Its behavior can be guided by physical interactions (e.g. sensor-actor coupling). This can be an efficient alternative to control using internal representations.
Social ability	An agent interacts and this interaction is marked by friendliness or pleasant social relations; that is, the agent is affable, companionable or friendly.
Sound	An agent is sound if it is predictive, interpretive and rational.
Stimulus response	A stimulus response agent has no internal state. It means that its responses are equal for equal inputs.
Temporal continuity	An agent is a continuously running process, not a "one-shot" computation that maps a single input to a single output, then terminates.
Transparency and accountability	An agent must be transparent when required, but must provide a log of its activities upon demand.
Trustworthiness	An agent adheres to laws of robotics and is truthful.
Unpredictability	An agent is able to act in ways that are not fully predictable, even if all the initial conditions are known. It is capable of nondeterministic behavior.
Veracity	It is the assumption that an agent will not knowingly communicate false information.

2.5. Multi-Agent Systems and Agent Communication

"*Distributed Problem Solving*" is performed by agents working together towards a solution of a common problem (e.g. for expert systems) [12]. *Multi-Agent Systems* (MAS) take a more general view of agents which have contact with each other in an environment (e.g. the Internet) [13]. The rules of the environment as well as the agent controls determine the form of *coordination*. The agents may be *cooperative* or *competitive*. Relations between local and global behavior in such MAS have been studied using game theory and social theories (cf. [101]).

Communication via exchange of messages is the usual prerequisite for coordination. Nevertheless, cooperation is possible even without communication, by observing the environment. The two most important approaches to communication are using protocols and using an evolving

language [28]. Both have their advantages and disadvantages. For industrial applications, communication protocols are the best practice, but in systems where homogeneous agents can work together, language evolution is the more acceptable option [28]. *Agent Communication Languages* (ACLs) provide important features like technical declarations (sender, receiver, broadcasting, peer-to-peer, …), speech act (query, inform, request, acknowledge,…) and content language (e.g. predicate logic). Together with these features, related protocols are defined to determine the expected reactions to messages (e.g. an inform message as an answer to query message). A number of languages for coordination and communication between agents was enumerated in [102]. The most prominent examples [102] are given in Table 2.

Table 2. Languages for coordination and communication between agents.

Agent communication language	***Description***
KQML ("Knowledge Query and Manipulation Language")	It is perhaps the most widely used agent communication language [102], [45]. KQML uses speech-act performatives such as reply, tell, deny, untell, etc. Every KQML message consists of a performative and additional data written in several slots. Some slots are :content, :in-reply-to, :sender, :receiver, :ontology, etc. The set of performatives in KQML and their slots should be general enough to enable agent communication in every agent application. There are claims that there might be some problems with the semantics of performatives. Various agents may interpret the same performative in various ways.
FIPA-ACL ("FIPA Agent Communication Language")	It is an agent communication language that is largely influenced by ARCOL [102]. FIPA ACL has been defined by FIPA - Foundation for Intelligent Physical Agents. Together FIPA-ACL [47], ARCOL, and KQML establish a quasi standard for agent communication languages [102]. Syntax and semantics of FIPA ACL are very similar to the syntax and semantics of KQML. Time will show which one of these two standards will prevail.
ARCOL ("ARTIMIS COmmunication Language")	ARCOL has a smaller set of communication primitives than KQML, but these can be composed. This communication language is used in the ARTIMIS system [102].

KIF ("Knowledge Interchange Format")	This logic-based comprehensive language with declarative semantics has been designed to express different kinds of knowledge and meta-knowledge [102]. KIF is a language for content communication, whereas languages like KQML, ARCOL, and FIPA-ACL are for intention communication.
COOL ("Domain independent COOrdination Language")	COOL relies on speech-act based communication, aims at explicitly representing and applying coordination knowledge for multi-agent systems and focuses on rule-based conversation management (conversation rules, error rules, continuation rules, …) [102]. Languages like COOL can be considered as supporting a coordination/communication (or "protocol-sensitive") layer above intention communication.

Contract Net Protocols and *Blackboard Systems* are well understood mechanisms for organizing MAS. Contract Net Protocols organize the distribution of tasks to other agents by announcing tasks, receiving bids from other agents and choosing one of the bidding agents for execution. Blackboard systems provide a common active database (the blackboard) for information exchange.

MAS with many agents are often used for simulations to study *Swarm Intelligence* and for social simulations in the field of *Socionics.* Social simulations include simulations of financial markets, traffic scenarios, and social relationships. Swarm intelligence can lead to complex "intelligent" behavior which emerges from the interaction of very simple agents, e.g. in ant colonies or in trade simulations. Complex problems, e.g. the well-known travelling salesman problem can be solved with swarm techniques.

3. Languages for constructing Agent-based systems

An essential component of agent-based technology and implementation of agent-based systems is a programming language. Such a language, called an *agent-oriented* programming language, should provide developers with high-level abstractions and constructs that allow direct implementation and usage of agent-related concepts: beliefs, goals, actions, plans, communication etc.

Most agent systems are still probably written in Java and C/C++ [102]. Although traditional languages are not well-suited for agent systems, it is achievable to implement them in Pascal, C, Lisp, or Prolog languages [79]. Typically, object-oriented languages (Smalltalk, Java, or C++) are easier to use for realization of agent systems as agents share some properties with objects such as encapsulation, inheritance and message passing but also

differ definitely from objects vis-à-vis polymorphism [79]. Apart from these standard languages, several prototype languages for implementing agent-based systems have been proposed to support better realization of agent-specific concepts.

Devising a sound classification and analysis methodology for agent programming languages is a very difficult task because of the highly-dimensional and sometimes interdependent heterogeneous criteria that can be taken into account, e.g. computational model, programming paradigm, formal semantics, usage of mental attitudes, architectural design choices, tools availability, platform integration, application areas, etc. Therefore, here we take a more pragmatic approach by firstly proposing a light top-level classification that takes into account those aspects that we consider most relevant for agent systems, i.e. the usage of mental attitudes. According to this classification we find: agent-oriented programming (AOP) languages; belief-desire-intention (BDI) languages, hybrid languages (that combine AOP and BDI within a single model), and other (prevalently declarative) languages. Understanding the current state of affairs is an essential step for future research efforts in the area of developing agent-oriented programming languages.

Table 3 (at the end of the paper) brings summary and specific information for all agent languages presented in the paper that we managed to collect from different sources: Web Page, IDE, Implementation language, Agent platform integration, Applications, Paradigm, and Textbook.

3.1. Agent-oriented programming model

The term *Agent-oriented Programming* (AOP) was coined in [90] to define a novel programming paradigm. It represents a computational framework whose central compositional notion is an agent, viewed as a software component with *mental qualities*, *communicative skills* and a *notion of time*. AOP is considered to be a specialization of object-oriented programming (OOP), but there are some important differences between these concepts ([107], [90]). Objects and agents differ in their *degree of autonomy*. Unlike objects, which directly *invoke* actions of other objects, agents *express their desire* for an action to be executed. In other words, in OOP the decision lies within the requesting entity, while in AOP the receiving entity has the control over its own behavior, by deciding whether an action is executed or not. Also, agents can often have *conflicting* interests, so it might be harmful for an agent to execute an action request from another agent. An additional difference is *flexibility*. Agents often exhibit pro-active and adaptive behavior and use learning to improve their performance over time. *The thread of control* is the final major difference. While multi-agent systems are multi-threaded by default, there is usually a single thread of control in OOP.

An important part of the AOP framework, as described in [90], is a programming language. *Agent-orient Programming Language* (APL) is a tool that provides a high-level of abstraction directed towards developing agents

and incorporates constructs for representing all the features defined by the framework. Most of all, it should allow developers to define agents and bind them to specific behaviors [87]; represent an agent's knowledge base, containing its mental state; and allow agents to communicate with each other.

The AOP paradigm was very influential for the further development of agent programming languages, resulting in a number of languages.

AGENT0

AGENT0 (see Table 3) [90], [107] and [9], was the first agent-oriented programming language that has been developed, providing a direct implementation of the agent-oriented paradigm. Although being more of a prototype than a "real" programming language, it gives a feel of how a large-scale system could be built using the AOP concept.

In AGENT0, an agent definition consists of four parts: a set of capabilities (describing what the agent can do), a set of beliefs, a set of commitments or intentions, and a set of commitment rules containing a message condition, a mental condition and an action [107]. Agents communicate with each other through an exchange of messages which can be one of three different types: (1) a request for performing an action, (2) an "unrequest", for refraining from an action, and (3) an informative message, used for passing information. Usually, *requests* and *unrequests* result in agent's commitments being modified, while an *inform* message results in a change in agent's beliefs. Furthermore, a message can be private, corresponding to an internally executed subroutine, or public, for communication with other agents in the environment. These messages can alter agent's beliefs and commitments, i.e. its mental state. A crucial task is, therefore, to maintain the agent's mental state in a consistent form. As proposed in [90], there are three different ways of achieving this: 1) Using formal methods and mathematical logic; 2) Heuristic methods; 3) Making the language for mental space description as simple as possible, thus enabling trivial verification (the solution applied in AGENT0). Shoham [90] proposes the model for agent execution using a simple loop, which every agent regularly iterates: 1) Read the current messages, and, if needed, update set of beliefs and commitments; 2) Execute all commitments for the current cycle. This can result in further modifications of beliefs.

PLACA

Planning Communicating Agents - PLACA (see Table 3) is an improvement of the AGENT0 language, extending it with *planning facilities*, which significantly reduce the intensity of communication ([95], [9]). In PLACA, an agent doesn't need to send a separate message each time it requests another agent to perform some action. Instead, it can provide another agent with a description of the desired final state. After checking the rule conditions are satisfied, as

well as by using its planning abilities, the receiving agent presents to the sender a plan of actions to execute in order to reach the desired state. This means that the agents communicate requests for actions via high-level goals.

The logical component of PLACA is similar to that of AGENT0, but it includes operators for planning. Due to introduction of plans, mental categories and syntax in PLACA are a bit different than those in AGENT0. If a received message satisfies the message condition and if the current mental state of the receiving agent satisfies the mental condition, then the agent's mental state will be changed and the agent will send appropriate messages.

PLACA, as AGENT0, is an experimental language, not designed for practical use.

Agent-K

Agent-K (see Table 3) is another extension of the AGENT0 [35]. It replaces custom communication messages (i.e. *request*, *unrequest* and *inform*) with the standardized KQML. This improves the general interoperability of agents and enables them to communicate with different types of agents (that employ KQML as well). It doesn't, however, include the improvements brought by PLACA. Merging of the two concepts is achieved by modifying the AGENT0 interpreter to handle KQML messages. Since the interpreter is implemented in Prolog, an intermediate level was introduced to convert the Lisp-style format of KQML messages into an unordered Prolog list of unary predicates. In addition, this layer transforms textual parts of a KQML message into tokens that can be handled by the interpreter. The interpreter has been modified to include these changes and to allow multiple actions to occur at the same time, i.e. when there is a match between an incoming message and multiple commitment rules. Because of that, each Agent-K agent is a separate process with its own instance of the interpreter.

As an addition, Agent-K uses the KAPI[1] library for agent communication, which can transport KQML messages over TCP/IP and e-mail to remote systems. Although the integration with KQML should improve the interoperability of agents, this was not fully achieved [35] because Agent-K uses Prolog to encode agents' beliefs and commitments (thus restricting the communication to other Prolog-based agents only). Authors of the language propose another language for knowledge representation, (e.g. KIF), to be used.

MetateM

The Concurrent MetateM, currently called simply MetateM (see Table 3), [14], [15] is probably one of the oldest programming languages for multi-agent systems. It was based on the direct execution of logical formulae [49], [106].

[1] The KAPI library is provided by Jay Weber, EIT

MetateM has its roots in formal specification using temporal logic by bringing in the idea of executable temporal specifications. Therefore, it can be equally well described as a temporal logic programming language that is based on temporal rather than on first-order logic.

A MetateM agent program consists of a set of temporal rules that are built according to the paradigm: *declarative past and imperative future*. Intuitively this means that: (i) the conditional part of the rules is interpreted declaratively by matching it with the history of the agent execution, i.e. what is true in the current state of the agent and what was true in the past states of the agent, and (ii) the execution part of the rules represents the choices that the agent is facing in the next state, as well as in future states. So, intuitively, the execution of a MetateM program is in fact the process of building a concrete model of the program specification using a forward chaining algorithm.

The current implementation of MetateM [123] is based on Java and it supports asynchronous and concurrent execution of multiple agents that are able to exchange messages such that each message sent is guaranteed to arrive at a future moment in time. Moreover, MetateM supports a dynamic structuring of agents based on two sets of agents that are associated with each agent in the system: (i) the *content* set representing those agents that the current agent can control, and (ii) the *context* set representing those agents that can influence the current agent. This style of grouping allows efficient agent communication using multicast messages [50].

April and MAIL

Agent PRocess Interaction Language - April (see Table 3) [74] is a process-oriented symbolic language that was not designed specifically for agent-oriented programming, but rather as a general-purpose multi-process development tool. Nevertheless, it provides the necessary infrastructure for developing and employing simple agents. The main entity in an April system is a *process*, which represents an agent in the multi-agent paradigm. An agent is identified by its private or public handle. Private handles are accessible within the system only, while the public handles are available to agents in other systems as well. Public handles are registered in the system's *name server* and as such can be found from other systems connected to it. These inter-connected name servers allow one to build a global April application.

April has a simple communication infrastructure that uses TCP/IP and permits access to non-April based applications. Agents communicate by exchanging messages identified by their handles. If two agents send a message to a third agent, the April system cannot guarantee that they will arrive in the order of transmission, since there is no global synchronization clock. What can be assured is that if one agent sends *n* messages to another, they will arrive in the order they were sent, but it is not always possible to determine how much time an operation will take to execute. Therefore, "April is not particularly suitable for time-critical real-time applications" [74].

A powerful feature offered by April is *macro*, which gives developers the ability to define new language constructs, based on the existing ones. One of the main purposes of macros was to serve as tools for developing new, richer and more agent-oriented languages on top of April. Concepts as messages based on a particular speech-act, agent mobility, knowledge handling etc. can also be simulated [74]. Authors of April intended to include these extensions in a more developer-friendly manner and to create a new agent-oriented programming language called MAIL. MAIL as a high level language was intended for realization of many common MAS. First version of April and MAIL specification was the subject of an ESPRIT project and April had to serve as implementation language for MAIL, in fact as the intermediary between C and MAIL. MAIL was prototyped using IC-Prolog II (distributed logic programming system). Funding for the project was cancelled before it was implemented.

VIVA

VIVA (see Table 3) [98], an agent-oriented declarative programming language, was based on theory of VIVid agents introduced by same author. A VIVid agent is a software-controlled system with state expressed in a form of beliefs and intentions (as mental categories) and with behavior represented by action and reaction rules. Its basic functionality covered possibility to represent and perform actions in order to generate and execute plans. VIVA was in accordance with agent-oriented programming paradigm, but it was slightly conservative as it adopted as many concepts as possible from Prolog and SQL. The basic design principles of VIVA apart from conservativeness were scalability and versatility [98].

An agent specified in VIVA could run on a number of hosts with the same or different hardware/software architectures. The composition of MAS and the locations of participating agents had to be specified before a VIVA application could run.

The language was intended for general-purpose software agent programming, embedded systems and robots but has not fulfilled expectations of the authors to be widely used in MAS.

GO!

Multi-paradigm programming language GO! (see Table 3) [32] is conceptually similar to April. It combines OOP, concurrent, logic and functional paradigms into a single framework. Based on April, GO! brings following extensions: knowledge representation features of logic programming, yielding a multi-threaded, strongly typed and higher order language (in the functional-programming aspect) [21]. In inheritance from April, threads primarily communicate through asynchronous message passing. Threads, as executing action rules, react to received messages using pattern matching and pattern-based message reaction rules. A communication daemon enables threads in

different GO! processes to communicate transparently over a network. Each agent usually can encompass several threads directly communicating with threads in other agents. Threads within a single GO! process can also communicate by manipulating a shared cell or dynamic relation objects.

As a strongly typed language it can improve code safety as well as reduce the programmer's burden. New types and new data constructors can be easily added. The designers of the language have had in mind critical issues like security, transparency and integrity, in regards to adoption of the logic programming essence. Features of Prolog like the cut ('!') have been left out for obvious reasons. In Prolog, the same clause syntax is used for defining relations (declarative semantics), and for defining procedures (operational semantics). In GO!, however, behavior is described using action rules expressed in a specialized syntax.

3.2. BDI based languages

A significant and influential trend in designing agent programming languages stemmed from the success of the practical reasoning agent architectures, among which the most notable is probably the PRS – Procedural Reasoning System [55]. PRS became the first system embodying a belief, desire, and intention (BDI) architecture. Based on that, approximately around the same time with Shoham, Rao proposed the AgentSpeak(L) language [83]. AgentSpeak(L) employs the metaphors of belief, desire, and intention of the BDI architecture to shape the design of an innovative agent programming language. However, AgentSpeak(L) was only a proposal, while Jason programming language became in 2004 the first implementation of an interpreter for an extended version of AgentSpeak(L) [22], [121]. AgentSpeak(L) is often described as a BDI agent programming language, as it is assumed to convey the most important ideas of BDI agent architectures (including the PRS).

In this section we will present several important agent programming languages which support BDI architecture and belong to the hybrid paradigm.

AgentSpeak

The language was originally called AgentSpeak(L) (see Table 3), but became more popular as AgentSpeak. This term is also used to refer to the variants of the original language. The primary goal of the authors of AgentSpeak [100] was to join BDI architectures for agents and for object-based concurrent programming and to develop a language that would capture the essential features of both. They identified the primary characteristics of agents: complex internal mental state, proactive or goal-directed behavior, communication through structured messages or speech acts, distribution over a wide-area network, adequate reaction to changes in the environment,

concurrent execution of plans and reflective or meta-level reasoning capabilities.

The basic construct in AgentSpeak is an *agent family* and its purpose is analogous to a class in object-oriented languages. Each agent (an instance of an agent family) contains a *public* and a *private* area which are, respectively, offered to other agents or used for agent's internal purposes. An agent's behavior is described using three different concepts: *database relations*, *services* and *plans*. Agents execute actions in order to meet their own, or desires of other agents. For fulfilling its own desires, an agent uses a set of private services (inner *goals)*, while other agents can invoke its public services (corresponding to messages from other agents). In AgentSpeak there are three distinct types of services for different purposes:

- *Achieve-service*: used to achieve a certain state of the world
- *Query-service*: used to check whether something is true, considering the associated database
- *Told-service*: used to share some information with another agent.

Once a service has been invoked, an agent proceeds to execute it by the means of plans. Once a plan has been activated, its *goal statements* are executed. Upon successful execution of all goal statements, the reached state is assessed in order to make sure that the desired state of affairs has been achieved.

Agents communicate in AgentSpeak by exchanging messages, either asynchronously (default) or synchronously. A message can be sent to a specific agent or to an agent family, in which case it is forwarded to all instances of that family. If a message sent to another agent contains some information, but puts no obligation upon the receiving agent, it is called an *inform* speech-act. Otherwise, it's a *request*. In addition, a message can have a *priority* assigned to it, thus giving it an overall importance.

In recent times, a work on *Coo-AgentSpeak* has been published in [2]. It incorporates ideas presented in *Coo-BDI* [3] into AgentSpeak. Coo-BDI extends the standard BDI model with cooperation, allowing agents to exchange their plans for satisfying intentions.

Jason

Jason (see Table 3) is probably the first implementation of AgentSpeak(L) using the Java programming language and belongs to the hybrid agent paradigm [22]. The syntax of Jason exhibits some similarities with Prolog. However, the semantics of the Jason language is different and it is based on AgentSpeak(L). One strength of Jason is that it is tightly integrated with Java with the following immediate consequences: (i) the behavior of the Jason interpreter can be tailored using Java; (ii) Jason can be used to build situated agents by providing a Java API for integration with an environment model that is developed with Java; (iii) Jason has been integrated with some existing agent frameworks, including JADE [18], AgentScape [97], and Agent Factory [113].

AF-APL

Agent Factory Agent Programming Language - AF-APL (see Table 3) is the core of Agent Factory agent development environment. AF-APL is originally based on Agent-Oriented Programming [90], but was revised and extended with BDI concepts (hybrid paradigm). AF-APL is described as a "practical rule-based language" based on *commitment rules*. A commitment rule joins together three types of mental attitudes: *beliefs*, *plans*, and *commitments*. The syntax and semantics of the AF-APL language have been derived from a logical model of how an agent commits itself to a course of action [33], [85]. The semantics of AF-APL was formalized in Rem Collier's Ph.D. thesis using multi-modal first-order branching-time logic [33].

An AF-APL programmer can declare explicitly, for each agent, a set of sensors (*situated* agents) referred to as perceptors and a set of effectors (actuators). Perceptors are in fact instances of Java classes which define how to convert raw sensor data into beliefs. An actuator is realized as an instance of a Java class with responsibilities: 1) to define the action identifier that should be used when referring to the action (realized by the actuator); 2) to contain code that implements the action. These declarations, specified within the agent program, are termed the embodiment configuration of the agent.

The AF-APL programming language is strongly related to the Agent Factory framework for the development and deployment of agent systems (see [76] for a recent overview and applications of Agent Factory framework).

3APL

3APL (see Table 3) [60] is not explicitly declared a descendant of either AgentSpeak(L), or Agent0. However, in our opinion it was clearly influenced by both AOP and BDI families of languages, and more important, both families of languages were clearly influenced by the general settings of the *intentional stance* towards understanding and development of a software system [38]. It is interesting to note that 3APL was theoretically shown to be at least as expressive as AgentSpeak(L) [59]. However, although it's Web page is still alive [111], we have noticed that 3APL language and supporting tools do not seem to be further developed. Rather, one of its authors, Koen V. Hindriks switched to the development of a new language GOAL. Nevertheless, 3APL is still relevant as it has opened the new direction of *goal-oriented agent-programming languages* and in some sense it has unified ideas from AOP, BDI and logic within a single programming model with declarative goals (hybrid paradigm). Moreover, there is an explicitly declared successor of 3APL called 2APL that is currently being developed [133]. 3APL has been applied to robot control using an API called ARIA (provided by ActivMedia Robotics[2]).

[2] http://www.activmedia.com/

2APL

2APL (see Table 3) is the successor of 3APL, enhancing it in many aspects. Probably the most important aspect is the clear separation of multi-agent and individual agent concerns. The multi-agent part is addressing the specification of a set of agents, a set of external environments and the relations between them, i.e. agent – agent and agent – external environment relations. The individual agent concepts in 2APL cover beliefs, goals, plans, events, messages, and rules, so it has many similarities with the programming notions that are available in other BDI and AOP languages. 2APL amalgamates declarative and imperative programming styles, so it can be described as hybrid (in the sense of the classification from [21]). Probably this is the most notable difference between 2APL and GOAL, as GOAL is clearly a declarative programming language, while 2APL is described by its authors as a "practical agent programming language". 2APL has been designed to work with JADE and, in comparison with 3APL, it provides practical extensions that allow better testing and debugging [34].

JACK Agent Language

JACK™ Intelligent Agents, or simply JACK (see Table 3), is a commercial agent platform provided by Autonomous Decision Making Software – AOS [130]. The main JACK components are: *JACK Agent Language* (also known as JAL), *JACK compiler*, *JACK kernel*, and *JACK Development Environment*. JAL is a superset of Java that incorporates the full Java language and provides the necessary constructs for building agent-oriented programs according to the BDI model. JAL is translated into JAVA source code using the *JACK compiler*, and the resulting Java code can be run on top of the JACK runtime engine, also known as *JACK kernel*. JACK Development Environment is an integrated graphical environment for the development of JACK multi-agent applications.

JACK supports the development of distributed agent applications by allowing agents to be deployed in separate processes, possibly running on different networked machines. JACK agents are able to exchange messages in a peer-to-peer fashion, as well as they are able to find each other using name servers. JACK and supporting tools are reviewed in [103].

JADEX

JADEX (see Table 3) is a Java-based agent platform that tries to respond to three categories of requirements: openness, middleware, and reasoning, thus bridging the gap between middleware-centered and reasoning-centered systems [82], [120]. The architecture of a JADEX agent follows the Procedural Reasoning System (PRS, [55]) computational model of practical reasoning. Agents in JADEX communicate by exchanging messages. Internally, an agent

reacts to events in its execution cycle that combines reaction and deliberation processes.

A JADEX agent uses the concepts of BDI agents: beliefs, desires (goals in JADEX), and intentions (plans in JADEX). JADEX employs an object-oriented representation of beliefs. Additionally, beliefs have an active role, i.e. their update can trigger generation of events or adoption/dismissing of goals. JADEX uses four types of goals: (i) *perform* goal designating action execution; (ii) *achieve* goal designating a point-wise condition in the lifecycle of an agent that must be reached; (iii) *query* goal that is an introspection mechanism by which an agent is inspecting its own internal state; (iv) *maintain* goal designating a process-wise condition that must be maintained during the agent's execution. JADEX plans represent the behavioral aspect of an agent and they have a procedural flavor. A plan consists of a *head* and a *body*, similarly to a procedure in a procedural language.

The JADEX language combines the declarative specification of an agent containing its set of beliefs, goals and plans using an Agent Definition File (ADF) and the procedural specification of the plan bodies using the Java programming language. The plan body accesses the internals of an agent through a specialized API. JADEX agents are able to run on the JADE middleware platform, thus enabling the development of distributed intelligent systems using the BDI metaphor.

3.3. Other Agent Languages

Within the generic class of "other languages" we include all those agent programming languages that do not explicitly employ mental attitudes for shaping the language, but rather use other constructs that are very useful for building intelligent software agents by supporting reasoning tasks based on formal logic, methods and calculi set on top of the main characteristics attributed to agents. Compared to AOP and BDI, this category can be characterized as a more traditional to agent programming from the point of view of computer science practices.

During the period of developing different agent-oriented programming languages, some authors and research groups proposed and implemented languages essentially based on and characterized as the declarative paradigm. In this section we will present several important agent programming languages which support the declarative paradigm.

GOAL[3]

The main motivation behind the development of Goal-Oriented Agent Language, i.e. GOAL (see Table 3) was to bridge the gap between agent logics and agent programming models (BDI and AOP) [61]. This new language introduces a declarative perspective of goals in agent programming languages by unifying the concepts of commitments from Agent0, intentions from AgentSpeak(L) and goals from 3APL. An interesting feature of GOAL is that it sets on a clean and unified theoretical basis the concepts of reasoning and knowledge representation from AI with the mentalist notions that are more specific to agent programming. The GOAL agent programming language was recently overviewed in [62]. According to this reference, GOAL has been tested on top of JADE. However, we were not able to find any references to such an experiment. The current implementation of GOAL [132] is just a prototype that is currently mainly used for educational purposes. However, it can be also useful in planning applications, for example in the transportation and logistics domain.

Golog

”alGOl in LOGic” – GOLOG (see Table 3), is a family of logic languages (declarative paradigm) based on the formalism of *situation calculus* that was developed in AI by John McCarthy for the specification of dynamic systems [75]. Situation calculus is a first-order logic language with some second-order extensions that utilizes the following concepts: (i) action; (ii) situation; and (iii) fluent. Changes in the world are modeled using the *action* concept. Histories of the world are modeled using the *situation* concept; a situation is in fact a sequence of actions. *Fluents* represent relations and functions that depend on the situation, thus we have *relational fluents* and *functional fluents*.

According to [118], the GOLOG family comprises the following languages: (i) GOLOG, the core language, initially introduced in [84]; (ii) ConGOLOG, i.e. Concurrent GOLOG, an extension of GOLOG for handling concurrency [37]; (iii) IndiGOLOG: Incremental deterministic GOLOG [37].

Recently, it was shown that the BDI-style of agent programming can be achieved with GOLOG [88], thus bridging the gap between BDI and action logic styles of agent programming.

In our literature review we have found that GOLOG was quite influential in the area of programming physical robots endowed with cognitive capabilities. This trend spawned a number of extensions of GOLOG. ICPGOLOG is an

[3] Note that the GOAL agent programming language developed by Koen V. Hindriks is not the same thing as the GOAL agent programming language proposed by (Byrne and Edwards, 1996) in Byrne, C. ; Edwards, P.: Refinement in Agent Groups. In: Weiß, G. ; Sen, S. (Eds.): Proceedings of the IJCAI’95 Workshop on Adaption and Learning in Multi-Agent Systems, Lecture Notes in Computer Science 1042, Springer, 1996, pp. 22–39. Byrne’s GOAL is a direct descendant of Agent-0 and it was proposed earlier than Hindriks’s GOAL.

extension of GOLOG with actions to describe continuous change, support for noisy sensors and effectors, and probabilistic actions [39]. Implementation of ICPGOLOG was based on the existing implementation of IndiGOLOG in Prolog. READYLOG is a robot programming and planning language that adds to GOLOG the logic specification of MDP theories for decision-theoretic planning [43]. A novel prototype implementation of the GOLOG interpreter using the Lua scripting language for the bi-ped robot platform Nao was recently reported in [44].

Although one can notice that the main focus of GOLOG was to model single robotic agents, there were also works that propose GOLOG extensions for multi-agent systems in a game-theoretic setting, namely GTGOLOG [46].

FLUX

Fluent executor – FLUX (see Table 3) is a logic programming language (declarative paradigm) based on *fluent calculus* [93]. Fluent calculus is an axiomatic theory of actions that represents an improvement of situation calculus [75], since in fluent calculus situations represent state descriptions, while in situation calculus they represent histories of action occurrences. Thus, FLUX has a declarative semantics. The language is extensively described in the textbook [94]. An important difference between FLUX and many other agent programming languages is that the main focus of FLUX is on programming single agents that act logically in a dynamic environment, rather than developing complex multi-agent systems. In this respect, FLUX is similar to GOLOG. There are works, however, that describe a practical multi-agent system that contains a set of agents, each one equipped with a FLUX interpreter, that cooperate to solve a complex problem [89]. The current implementation of FLUX is based on constraint logic programming systems (Eclipse Prolog and Sicstus Prolog) for efficient handling of the axioms of fluent calculus.

CLAIM

Computational Language for Autonomous, Intelligent and Mobile agents – CLAIM (see Table 3) is a high-level agent programming language that combines the basic functionalities required for the agent model with higher-level support specific to intelligent and cognitive abilities (belongs to the hybrid paradigm). An important characteristic of CLAIM is its built-in support for agent mobility that is based on the abstract computation model of *ambient calculus* [30]. CLAIM agents are hierarchically structured (according to the formal model of ambients), goal-directed, knowledge-based, able to communicate at knowledge level, and mobile. CLAIM agents are not entirely declarative, as they mix declarative characteristics required for the specification of the knowledge component with imperative capabilities, required for the specification of the capabilities component. CLAIM is part of

Hymalaya unified framework and it is supported by SyMPA distributed multi-agent platform. Unfortunately, there is not much information about any of them, excepting the research papers [91] and [41].

4. Tools and Platforms

Multi-agent systems are deployed and run over specialized software infrastructures that provide the set of functionalities vital for the existence of a realistic multi-agent application. Seen from the perspective of distributed systems technologies, such infrastructures are placed at the middleware level and they include a collection of software functionalities and services that assure: agent lifetime management, agent communication and message transport, agent naming and discovery, mobility, security, etc. An *agent framework* is a software infrastructure available as a software library, a language environment, or both, which provides the core software artifacts needed for creating the skeleton of a multi-agent system. A software package that provides the core functionalities for deploying and running multi-agent applications is traditionally known as an *agent platform* [96]. An *agent toolkit* is a more complex software infrastructure that allows both the development and deployment of a multi-agent system [69]. It is sometimes known as an *agent development environment* [96], because of its expected support for all engineering stages of a multi-agent application from requirements to deployment, maintenance and evolution.

Most often, a multi-agent system is deployed and runs on top of an agent platform. If an agent platform is not available, at least an agent framework is usually utilized to create the multi-agent system which is then run on a general purpose middleware platform. Agent code can be programmed either using a general-purpose programming language linking with software libraries available in the agent framework via the framework API, or using one of the agent programming languages (see the previous section).

Agent platforms can be extremely useful because they considerably simplify the development and deployment of a multi-agent system. There is the option to choose between standardized or not-standardized agent platforms. A standard agent platform is compliant with available standards for software agents. Compliance to standards is important for open systems, i.e. systems that might need to interoperate in the future with other systems that are either not available at the moment when the open systems are being developed or that, even if they are available at the moment, still might change in the future.

According to our literature review, more than 100 agent platforms and toolkits were developed (or started to be developed) [69] of different quality and maturity. Most of them are built on top of and are integrated with Java [102]. Despite this fact that clearly shows that software agent technologies triggered a significant initial interest and hope, only few of them are still currently available, while the rest either became obsolete or are not being

developed anymore. In the rest of the section, we provide a brief review of some of them. Although our selection might look quite subjective, we have done our best to consider those agent platforms and toolkits that we think are most influential, currently active and also well supported by the open source and/or business communities. Note that some of the platforms considered in this paper are also overviewed in more detail by [96].

4.1. ZEUS

ZEUS [129], [4], [81], [80] developed by British Telecommunications Labs, is a collaborative agent building environment that has excellent GUI and debugging, provides library of predefined coordination strategies, general purpose planning and scheduling mechanism, self-executing behavior scripts, etc. ZEUS is one of the most complete and the most powerful agent tools which are used to design, to develop and to organize agent systems. The aim of ZEUS project was to facilitate the rapid development of multi-agent applications by abstracting into a toolkit the common principles and components underlying some existing multi-agent systems [78]. It enables applications with additional assistant tools, e.g. reports and statistics tools, agents and society viewer, etc. ZEUS documentation is very weak, which leads to difficulties in creating new applications. The three main functional components of ZEUS are (adapted from [129]): The Agent Component Library; The Agent Building Tools; The Visualization Tools.

Some characteristics of ZEUS are: it implements FIPA standards, supports KQML and ACL communication and security policy supports ASCII-encoded, Safe-Tcl scripts or MIME-compatible e-mail messages for transportation; it uses public-key and private-key digital signature technology for authentication, cash and secrecy.

4.2. JADE

Java Agent DEvelopment Framework - JADE is probably one of the most popular agent platforms that are currently available to the open source community. JADE is FIPA-compliant and it is well supported by documentation [119], a textbook [18] and an enthusiastic community of users.

A JADE agent platform can be distributed on multiple machines that run the Java virtual machine, while multiple platforms can interoperate via FIPA standards. A platform consists of multiple containers, while each container can contain zero or more JADE agents. There is exactly one *Main* container and, optionally, zero or more ordinary containers, linked to the *Main* container. The JADE containers can be distributed onto the nodes of a local area network. Each node can host several containers. Each JADE agent contains its own execution thread. Unfortunately, this design choice is one of the main limitations for the number of agents that can be created and executed on a

single machine. JADE agents use a specialized execution model based on non-preemptive scheduling of dynamically loadable JAVA plugins called *behaviors*. The agent execution model combined with JADE's intuitive programming interface allows the programmer to relatively easily develop software agents that are capable of flexible reactive and/or proactive behaviors. JADE agents can interact by asynchronously exchanging FIPA ACL messages, optionally following FIPA interaction protocols [116]. Telecom Italia is currently used JADE as reference framework for Network Neutral Element Manager – NNEM project [19].

4.3. agenTool

agenTool is a Java-based graphical development environment/tool that supports the Multi-agent Systems Engineering (MaSE) methodology [39] originally developed at the Artificial Intelligence Lab of the Air Force Institute of Technology, Ohio. It implements all MaSE steps including conversation verification and code generation. One of its most interesting abilities is the possibility to work on different pieces of the system and at various levels of abstraction interchangeably, which mirrors the ability of MaSE to incrementally add detail [39]. During each step of system development it is possible to use various analysis and design diagrams. . Moreover, it is possible to transform a set of analysis models into appropriate design models using semi-automatic transformations. Some efforts have been done in order to support modeling of mobile agents.

4.4. RETSINA

Reusable Environment for Task-Structured Intelligent Networked Agents – RETSINA is a multi-agent system toolkit that has been developed since 1995 at the Intelligent Software Agents laboratory of Carnegie Mellon University's Robotic Institute [125].

RETSINA is probably one of the earliest, most influential software infrastructures for developing multi-agent systems. It supports the development of communities of heterogeneous agents that can engage in peer-to-peer relations without imposing any centralized control for agent management. A RETSINA-based multi-agent system is platform independent, being able to run on various operating systems, while its agents can be implemented using different general-purpose programming languages. RETSINA is using a multi-agent software infrastructure based on Agent Foundation Classes – AFC. A very good overview of the distributed software infrastructure of RETSINA is provided by [92].

RETSINA was utilized for developing an impressive number of applications in various areas: military operations, critical decision making, supply chain management, financial portfolio management, text mining, etc [125], [92].

4.5. JATLite

'Java Agent Template, Lite' - JATLite [65] has been developed at the Stanford Center for Design. The intention was to allow creating software *typed-message agents* communicating over the Internet. Agents communicated using typed messages in an agent communication language like KQML, in which some semantics are defined before runtime. Two additional requirements had to be fulfilled: *Reliable message delivery and Migrating agent communication*.

JATLite added basic infrastructure functionality that earlier systems missed, supporting buffered-message exchanges and file transfers with other agents on the Internet, as well as connection, disconnection, and reconnection in the joint computation [65]. Security aspects of JATLite message relied on current open standards for encryption and authentication. The one simple feature that JATLite added was a password associated with the agent name.

JATLite featured modular construction consisting of increasingly specialized layers: protocol, Router, KOMC, Base and Abstract layer. Developers could select the appropriate layer to start building their systems. Each layer could be exchanged with other technologies without affecting the operation of the rest of the package.

4.6. FIPA-OS

FIPA-OS [117] is a component-based toolkit enabling rapid development of FIPA compliant agents. It was first released in August 1999 supporting the majority of the FIPA specifications. It has been continuously improved until 2003 and was publicly available as an ideal choice for FIPA compliant agent development. There have been two versions of FIPA-OS:

- **Standard FIPA-OS** - Two alternative distributions were provided: Java 2 (JDK1.2) compatible version (containing code developed directly from the FIPA-OS codebase) and Java 1.1 compatible version (containing code, which has undergone automated code-refactoring to enable the JDK1.2 compatible code of FIPA-OS to be used with JDK1.1).
- **MicroFIPA-OS** - This is an extension to the JDK 1.1 version of FIPA-OS and has been designed to execute on PDAs (that can execute a PersonalJava compatible virtual machine).

Both FIPA-OS versions use tasks and conversations as the basis for support to agents' functionalities. Developers using FIPA-OS have been encouraged to provide extensions, bug fixes and feedback to help improve different releases.

4.7. MADKIT

Multi-agent development kit - MadKit [122] [56] is an open source modular and scalable multi-agent platform which has been developed at LIRMM (France), built upon the AGR (Agent/Group/Role) organizational model (Aalaadin [42]). MadKit is written in Java and MadKit agents play roles in groups and thus create artificial societies. In addition to AGR concepts, the platform adds three design principles: Micro-kernel architecture; Agentification of services; Graphic component model.

The last version was released in November 2010. MadKit is a set of packages of Java classes that implements the agent kernel, various libraries of messages, probes and agents. This platform is not a classical agent platform as any service, besides those assured by micro-kernel, is handled by agents. Micro-kernel and existence of a range of modular services managed by agents enable a range of multiple and scalable platforms. Communication is achieved through asynchronous message passing: 1) by primitives used to send a message directly to another agent represented by its *AgentAddress*, or 2) by higher-level functions that send or broadcast to one or all agents having a given role in a specific group. MadKit uses agents to achieve distributed message passing, migration control, dynamic security, and other aspect of system management.

MadKit has been used in various projects covering a wide range of applications [67], from simulation of hybrid architectures for control of submarine robots to evaluation of social networks or study of multi-agent control in a production line.

4.8. JAFMAS

Java-based Agent Framework for Multi-Agent Systems - JAFMAS [36], is a framework for representing and developing cooperation knowledge and protocols in a multi-agent system (coordinating their knowledge, plans, and goals so that they can take actions which result in coherent joint problem solution). This framework provides a generic methodology for developing speech-act based multi-agent systems and follows several stages: agent identification, definition of each agent's conversations, determining the rules governing each agent's conversations, analyzing the coherency between all the conversations in the system, and implementation. JAFMAS provides communication (directed and subject-based broadcast), linguistics for speech-acts (e.g. KQML) and coordination support. Such functionality is based on COOL (coordination Lisp-based language for explicitly representing, applying and capturing cooperation knowledge for multi-agent systems). In COOL and JAFMAS, an agent is a programmable entity that can exchange messages within structured "conversations" with other agents, change state and perform actions. JAFMAS agents support conversation based on message exchange according to mutually agreed conventions, change state and perform local

actions. Different researchers still use JAFMAS framework for developing multi-agent systems [108] [99].

4.9. Agent Building Shell

Agent Building Shell - ABS was developed at University of Toronto [10]. ABS provides several reusable layers of languages and services for building agent systems. The layers of the architecture achieve a range of functionalities [112]. The shell supports KQML/KIF based communication. COOL is provided and built on top of the agent communication language. The language supports definition, execution and validation of complex speech-act based cooperation protocols. Multiple, parallel conversations are possible and their management can be programmed through specific control mechanisms. Interaction between users (using web browsers) and agents is conversation based, using the same conversational infrastructure that supports interactions among agents. Agents negotiate by exchanging constraints about the performance of activities. In the negotiation process, agents send their requests to other agents and receive either confirmations or explanations why their requests cannot be satisfied. Agents employ a unified behavior description language that specifies behaviors as consisting of sequential, parallel and choice compositions of actions. Specific constraint propagation mechanisms are used to determine which actions will be executed. At the organization level, agents acquire authority to make requests and impose violation costs from the roles they play in the organization. Concerning knowledge management, there is a representational substrate that provides services for carrying out the various reasoning tasks outlined.

According to several authors [51], [11], [71], ABS has been considered appropriate for developing agents in supply chain management systems.

4.10. OAA

Open Agent Architecture – OAA [124] was developed in Artificial Intelligence Center, California and its last version was released in 2007. It is a framework for integrating a community of heterogeneous software agents in a distributed environment. OAA facilitates flexible, adaptable interactions among distributed components through delegation of tasks, data requests and triggers; and enables natural, mobile, multimodal user interfaces to distributed services. OAA is structured to minimize the effort in creating agents and "wrapping" legacy applications, written in various languages and platforms; to encourage the reuse of existing agents; and to allow for dynamism and flexibility in the makeup of agent communities. Unique features of OAA include great flexibility in using facilitator-based delegation of complex goals, triggers, and data management requests; agent-based provision of multimodal user interfaces; and built-in support for including the user as a privileged member of the agent

community. The system has been used in different applications and some of them are:

- framework of transformer condition assessment system employing data warehouse, data mining, and Open Agent Architecture [68].
- multi-agent architecture with distributed coordination for an autonomous robot [5].

4.11. Cougaar

Cognitive Agent Architecture – Cougaar [115] is an open-source Java-based agent platform developed as result of a multi-year project of DARPA research. Cougaar is not FIPA-compliant, and more important, it was not designed for standards compliance. Cougaar agents are composed of plugins that communicate sharing common and distributed data space - *blackboard architecture*. Agents can subscribe for automatically receiving blackboard updates. The plugins communicate by publishing (adding) new objects to the blackboard, making changes to objects already published or removing objects from the blackboard. When special objects called *relays* are published onto the blackboard they are automatically forwarded by the blackboard system to other agents, thus achieving the communication between agents.

The main focus of its development was scalability [57] and as a consequence it was mostly utilized for the development of applications in military logistics [31].

4.12. AgentScape

AgentScape was developed at Delft University of Technology as a middleware platform that provides a minimal set of concepts and functionalities for the development of large-scale distributed multi-agent systems. The focus in AgentScape was set on: (i) scalability; (ii) heterogeneity through multiple code bases, programming languages and operating systems; (iii) interoperability [114]. Although AgentScape is a very interesting platform, it currently suffers from the problem that the documentation is not mature enough and is rather incomplete. Nevertheless, AgentScape has been applied in a number of interesting research and commercial projects related to the electricity market [17] and e-commerce [40].

4.13. Cybele

Cybele™ is a commercial agent platform provided by Intelligent Automation Inc. for the development and deployment of large-scale distributed intelligent systems [131]. Cybele™ is built on top of Java platform. Agents are programmed in Java using a standard style of programming called Activity

Centric Programming (ACP). This means that the basic building blocks of an agent are *activities*, while accesses to the basic functionalities of Cybele™ are provided via an Activity Oriented Programming Interface (AOPI). Cybele™ allows the development of distributed applications by installing it on several (at least 2) network nodes that together define a Cybele™ community. Exactly one node is designated as a *master node*, while the rest of them are *slave nodes*. A Cybele™ node can host several specialized Java applications known as Cybele™ *containers*. A container provides the runtime environment for a set of Cybele™ agents. It is not difficult to observe that an activity in Cybele™ has similarities with behavior in JADE, as well as with a plugin in Cougaar. Moreover, the method of structuring a distributed agent application into nodes, containers, agents and activities / plugins / behaviors is also used by JADE and Cougaar.

Cybele™ can be utilized as a platform for distributed robotics. The Distributed Control Framework (DCF) is a framework for building robotics applications for robot team coordination and management. Cybele™ is used as a core for DCF which supports two types of robotic agents: (i) *Robot Agent* that embodies a real or a simulated robot; (ii) *Remote Control Agent* that provides the control interface for a human operator with a robot team. Additionally, DCF includes a suite of components for sensing, estimation and control of several commercial robotic platforms.

5. Conclusion

Software agents are an emergent and rapidly developing field of research. In the last decade, a number of essential advances have been made in the design and development of software agent languages and the implementation of multi-agent systems. In this brief survey, we have tried to bring some of the key concepts, languages, tools and platforms and make a reference point to a large body of literature. Our intention was to enumerate and present essential features and functionalities of selected languages, tools and platforms, instead of judging them.

We consider an orthogonal classification by looking at the way agent programming languages are used during the systems development process. On one side, we can find agent languages useful for building software agents that can be used as building blocks for the development and deployment of complex distributed applications, usually based on agent or other suitable middleware platforms. On the other hand, we can find agent programming languages used for designing and running complex simulation models that employ the agent metaphor for modeling and simulation of complex systems. However, these languages are not immediately useful for developing real systems, but are rather mostly employed for research in understanding complex systems using agent-based modeling and simulation tools, as agent simulation languages. Note that this class of languages is very often forgotten by the existing works that overview advances in agent programming.

Nevertheless, between the two extremes we can find agent languages that are useful for both systems simulation, as well as for systems development and deployment.

In Table 3 we give a brief summary of agent programming languages. It can be noted that almost all of them, particularly the recently developed ones, have appropriate web-sites and IDEs. Despite the fact that there are representatives of different programming paradigms (imperative, declarative, BDI, hybrid), almost all of them are implemented in Java and a significant number of them are implemented in Prolog. Most of the recently developed languages find their place in real environments and have been used in developing different kinds of applications. Unfortunately, for majority of them there are no appropriate textbooks.

Note that we were able to find in the literature other overview works that provide classifications and comparisons of agent programming languages. Authors of [21] propose a classification of agent programming languages based on a lightweight interpretation of the programming paradigm as imperative, declarative, and hybrid (i.e. between declarative and imperative).

For the development and deployment of a multi-agent system in real environments it is necessary that appropriate software infrastructures (frameworks, tools, platforms) exist.

According to our literature survey, more than 100 agent infrastructures have been developed in the previous two decades. For portability and usability reasons most of them are built on top of and are integrated with Java [102]. Unfortunately, only few of them are still currently available, others either becoming obsolete or not being developed anymore.

Futhermore, this prominent technology inspired some authors to go a step further. In [70] authors extrapolated future trends in multi-agent systems and presented a thorough and outstanding approach to the future of multi-agent systems. Finally, it is important to mention that in order to be accepted by the industrial community, MAS applications need to be successfully demonstrated in complex real world pilot systems [29].

Acknowledgment. Research was partially supported by the Ministry of Education and Science, Republic of Serbia, through project no. OI-174023 'Intelligent techniques and their integration into wide-spectrum decision support'.

References

1. Agents mailing list, agents@cs.umbc.edu.
2. Ancona, D., Mascardi, V., Hübner, J.F., Bordini, R.H.: Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange, In Third International Joint Conference on Autonomous Agents and Multiagent Systems, Vol. 2, pp. 696 – 705 (2004)
3. Ancona, D., Mascardi, V.: Coo-BDI: Extending the BDI Model with Cooperativity, In Declarative Agent Languages and Technologies, Vol. 2990, pp.109-134 (2004)
4. Azarmi, N., Thompson, S.: "ZEUS: A Toolkit for Building Multi-Agent Systems", Proceedings of fifth annual Embracing Complexity Conference, Paris, (2000)

5. Badano B.M.I., A multi-agent architecture with distributed coordination for an autonomous robot, PhD theses, University of Girona, (October 2008)
6. Badjonski, M., Ivanović, M.: "Multi-agent System for Determination of Optimal Hybrid for Seeding", Proceedings of EFITA '97 - First European Conference for Information Technology in Agriculture, Copenhagen, Denmark, June 15-18, pp. 401-404. (1997)
7. Badjonski, M., Ivanović, M., Budimac, Z.: "Possibility of using Multi-Agent System in Education", Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Orlando, Florida, USA, October 12-15, pp. 588-593. (1997)
8. Badjonski, M., Ivanović, M., Budimac, Z.: "Software Specification Using LASS", Proceedings of Asian'97, Lecture Notes in Computer Science Vol 1345, Springer-Verlag, Kathmandu, Nepal, pp. 375-376. (1997)
9. Badjonski, M.: Adaptable Java Agents – a Tool for Programming of Multi-Agent Systems, PhD thesis, Department of Mathematics and Informatics, Faculty of Natural Science, University of Novi Sad (2003)
10. Barbuceanu, M., Fox, M.S., The architecture of an agent building shell. Intelligent Agents II, LNAI 1037, Spinger-Verlag, pp. 235-250 (1996)
11. Fox M.S., Barbuceanu M., Teigen R.: Agent-Oriented Supply Chain Management, International Journal of Flexible Manufacturing System, vol 12, pp. 165-188. (2000)
12. Bădică, C., Manufacturing and Control: Putting Agents to Work, IEEE Distributed Systems Online, vol. 8, no. 6, pp. 5, (2007)
13. Bădică, C., Ganzha, M., Paprzycki, M.: Developing a Model Agent-based E-Commerce System. In: Jie Lu, Guangquan Zhang, and Da Ruan (eds.): E-service Intelligence, Studies in Computational Intelligence, Volume 37, Springer, 555-578 (2007)
14. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: METATEM: A Framework for Programming in Temporal Logic, In: Proceedings on Stepwise refinement of distributed systems: models, formalisms, correctness, REX workshop, LNCS Volume 430, pp. 94-129 (1990)
15. Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R.: METATEM: An introduction. Formal Aspects of Computing 7(5), pp. 533–549 (1995)
16. Bratman, M.E.: Intention, Plans and Practical Reason. Harvard University Press, 1987.
17. Brazier, F., Cornelissen, F., Gustavsson, R., Jonker, C.M., Lindeberg, O., Polak, B., Treur, J.: A Multi-Agent System Performing One-to-Many Negotiation for Load Balancing of Electricity Use. In: Electronic Commerce Research and Applications Journal, vol.1, no.2, pp. 208-224, Elsevier, (2002)
18. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE, John Wiley & Sons (2007)
19. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE: A software framework for developing multi-agent applications. Lessons learned, Information and Software Technology, Volume 50, Issues 1-2, Elsevier, pp. 10-21. (2008)
20. Bordini, R.H., Dix, J., Dastani, M., Seghrouchni, A.E.F.: Multi-Agent Programming Languages, Platforms and Applications, Springer, (2005)
21. Bordini, R.H., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A Survey of Programming Languages and Platforms for Multi-Agent Systems, Informatica, no.30, pp. 33-44 (2006)
22. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason, John Wiley & Sons, (2007)
23. Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F. (Eds.): Multi-Agent Programming: Languages, Tools and Applications, Springer (2009)

24. Brooks, R.A.: "A Robust Layered Control System for a Mobile Robot", IEEE Journal of Robotics and Automation, 2(1), pp. 14-23. (1986)
25. Brooks, R.A.: "Intelligence without Reason", Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, pp. 569-595. (1991)
26. Brooks, R.A.: "Intelligence without Representation", Artificial Intelligence, 47, pp 139-159. (1991)
27. Budimac, Z., Ivanović, M., Popović, A.: "Workflow Management System Using Mobile Agents", Proceedings of ADBIS '99, Lecture Notes in Computer Science, Maribor, Slovenia, pp. 169-178. (1999)
28. Bussink, D.: A Comparison of Language Evolution and Communication Protocols in Multi-agent Systems. 1st Twente Student Conference on IT, Track C - Intelligent_Interaction, http://referaat.ewi.utwente.nl/ (2004)
29. Camarinha-Matos, L. M.: Multi-agent systems in virtual enterprises. Proceedings of AIS'2002 – International Conference on AI, Simulation and Planning in High Autonomy Systems, SCS publication, Lisbon, Portugal, pp. 27-36. (2002)
30. Cardelli, L., Gordon, A.D.: Mobile ambients. Foundations of Software Science and Computational Structures, Lecture Notes in Artificial Intelligence 1378, Springer, pp. 140-155. (1998)
31. Carrico, T., Greaves, M.: Agent Applications in Defense Logistics. In: Defence Industry Applications of Autonomous Agents and Multi-Agent Systems, Whitestein Series in Software Agent Technologies and Autonomic Computing, Birkhäuser Basel, pp. 51-72. (2008)
32. Clark, K. L., McCabe, F. G.: Go! – A Multi-paradigm Programming Language for Implementing Multi-threaded Agents, In Annals of Mathematics and Artificial Intelligence, Vol. 41, Issue 2 – 4, pp. 171 – 206, (2004)
33. Collier, R.W.: Agent Factory: A Framework for the Engineering of Agent-Oriented Applications, Doctoral Thesis, University College Dublin, Ireland, (2001)
34. Dastani, M.: 2APL: a practical agent programming language, International Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), 16(3), pp. 214-248 (2008)
35. Davies, W.H.E., Edwards, P.: Agent-K: An Integration of AOP and KQML, In Proceedings of the Third International Conference on Information and Knowledge Management, ACM Press, (1994)
36. Chauhan, D., Baker, A.D.: JAFMAS: a multiagent application development system. In Proceedings of the second international conference on Autonomous agents (AGENTS '98), http://doi.acm.org/10.1145/280765.280782 (1998)
37. De Giacomo, G., Lespérance, Y., Levesque, H.J., Sardina, S.: IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents, In [23], Springer, pp. 31-72. (2009)
38. Dennett, D.: Intentional Systems. In: Journal of Philosophy No. 68, pp. 87–106. (1971)
39. Dylla, F., Ferrein, A., Lakemeyer, G.: Specifying multirobot coordination in ICPGolog - from simulation towards real robots. In Proc. of the Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating (IJCAI 03), (2003)
40. El-Akehal, E.E., Padget, J.: Pan-supplier stock control in a virtual warehouse. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems AAMAS '08, pp. 11-18 (2008)
41. Seghrouchni, A.E.F., Suna, A.: CLAIM and SyMPA: A Programming Environment for Intelligent and Mobile Agents. In: [20], pp. 95-122, Springer, (2005)

42. Ferber, J., O. Gutknecht: Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems, In Proceedings of the Third International Conference on Multi-Agent Systems, ICMAS'98 pp. 128-135. (1998)
43. Ferrein, A., Lakemeyer, G.: Logic-based robot control in highly dynamic domains, Robotics and Autonomous Systems, volume 56, issue 11, North-Holland Publishing Co., 980-991, (2008)
44. Ferrein, A.: golog.lua: Towards a Non-Prolog Implementation of Golog for Embedded Systems. In: Cognitive Robotics, Dagstuhl Seminar Proceedings, no.10081, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Also in Proceedings of AAAI Spring Symposium 2010 on Embedded Reasoning, Stanford University, (2010)
45. Finin, T., Weber, J., et. al.: "Draft Specification of the KQML Agent-Communication Language", The Darpa Knowledge Sharing Initiative External Interfaces Working Group, available as http://www.cs.umbc.edu/kqml/kqmlspec.ps. (1993)
46. Finzi, A., Lukasiewicz, T.: Game-Theoretic Agent Programming in Golog, In: Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'2004, pp. 23-27, IOS Press, (2004)
47. Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification", available at http://www.fipa.org/specs/fipa00061/
48. Fisher M.: "Representing and Executing Agent-Based Systems", Intelligent Agents, Lecture Notes in Artificial Intelligence, Vol. 890, Springer-Verlag, pp. 307-323. (1994)
49. Fisher, M.: A Survey of Concurrent MetateM – The Language and its Applications, In Proceedings of the First International Conference on Temporal Logic, LNCS, Vol. 827, pp. 480 – 505, (1994)
50. Fisher, M., Hepple, A.: Executing Logical Agent Specifications. In [23], pp. 3-29, (2009)
51. Forget P., D'Amours S., Frayret J.M.: Multi-Behavior Agent Model for Planning in Supply Chains: An Application to the Lumber Industry, Universite Laval, Quebec, Canada, Working paper DT-2006-SD-03, https://www.cirrelt.ca/DocumentsTravail/2006/DT-2006-SD-03.pdf (2006)
52. Franklin, S., Graesser, A.: "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", Working Notes of the Third International Workshop on Agent Theories, Architectures and Languages, ECAI '96, Budapest, Hungary, pp. 193-206. (1996)
53. Fuggeta, A., Picco, G.P.: Understanding Code Mobility, IEEE Transactions on Software Engineering, vol.24, no.5, pp.342-361, (1998)
54. Georgakarakou, C. E., Economides, A. A.: Software agent technology: An overview. In: Software Applications: Concepts, Methodologies, Tools, and Applications, P. F. Tiako (ed.), IGI-Global ISBN: 978-1-60566-060-8 (2007)
55. Georgeff, M., Lansky, A.: Reactive reasoning and planning. In: Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87), pp. 677-682, (1987)
56. Gutknecht, O., Ferber, J.: The MADKIT Agent Platform Architecture. Agents Workshop on Infrastructure for Multi-Agent Systems, (2000)
57. Helsinger, A., Thome, M., Wright, T.: Cougaar: A Scalable, Distributed Multi-Agent Architecture, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 1910 – 1917, vol.2, IEEE Computer Society Press, (2004)
58. Hewitt, C.: The Challenge of Open Systems. Byte Magazine 10, 4, pp. 223-242, (April 1985)

59. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J-J.C.: A formal embedding of AgentSpeak(L) in 3APL, Advanced Topics in Artificial Intelligence, LNCS 1502, pp. 155-166, (1998)

60. Hindriks, K.V., De Boer, F.S., van der Hoek, W., Meyer, J-J.C.: Agent Programming in 3APL, Autonomous Agents and Multi-Agent Systems Volume 2, Number 4, pp. 357-401, (1999)
61. Hindriks, K.V., De Boer, F.S., van der Hoek, W., Meyer, J-J.C.: "Agent Programming with Declarative Goals", Intelligent Agents VII. Agent Theories Architectures and Languages, LNCS 1986, pp. 248-257, Springer, (2001)
62. Hindriks, K.V.: Programming Rational Agents in GOAL. In [23], 119-157, Springer, (2009)
63. Huang, J., Jennings, N., Fox, J.: "An Agent Architecture for Distributed Medical Care", Intelligent Agents, Lecture Notes in Artificial Intelligence, Vol 890, Springer-Verlag, pp. 219-232. (1994)
64. Jennings, N.R., Wooldridge, M.: "Software Agents", IEE Review, January, pp. 17-20. (1996)
65. Jeon, H., Petrie, C., Cutkosky, M.R.: JATLite: A Java Agent Infrastructure with Message Routing, IEEE Internet Computing, pp. 2-11. (2000)
66. Kaelbling, L.P.: "A Situated Automata Approach to the Design of Embedded Agents", SIGART Bulletin, 2(4), pp. 85-88, (1991)
67. Kallel I., Chatty A., Allimi A.M.: Self-Organizing Multirobot Exploration through Counter-Ant Algorithm, Proceedings Self-Organizing Systems: Third International Workshop, Iwsos 2008, Vienna, Austria, December 10-12, 2008, Springer. (2008)
68. Wu, L., Yongli, Z., Yuan, J., Li, X.: "Application of Open Agent Architecture and Data Mining Techniques to Transformer Condition Assessment System," International Journal of Emerging Electric Power Systems: Vol. 2 : Iss. 1, Article 1034. (2005)
69. Luck, M., Ashri, R., d'Inverno, M.: Agent-Based Software Development, Artech House, 2004
70. Luck, M., McBurney, P., Gonzalez-Palacios, J: Agent-Based Computing and Programming of Agent Systems. LNCS, Agent-Based Computing and Programming of Agent Systems, 3862, pp. 23-37, Springer. (2006)
71. Madejski, J.: Survey of the agent-based approach to intelligent manufacturing, Journal of of Achievements in Materials and Manufacturing Engineering, VOLUME 21, ISSUE 1, pp. 67-70. (2007)
72. Maes, P.: "The Agent Network Architecture (ANA)", SIGART Bulletin, 2(4), pp. 115-120, (1991)
73. Maes, P.: "Modelling Adaptive Autonomous Agents", Artificial Life Journal, Ed. C. Langton, Vol 1, No. 1&2, MIT Press, pp. 135-162. (1994)
74. McCabe, F. G., Clark, K. L.: April – Agent PRocess Interaction Language, In Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents, pp. 324 – 340 (1995)
75. McCarthy, J.: Situations, actions and causal laws. Technical report, Stanford University, 1963. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., pp. 410-417 (1968)
76. Muldoon, C., O'Hare, G.M.P., Collier, R.W., O'Grady, M.J.: Towards Pervasive Intelligence: Reflections on the Evolution of the Agent Factory Framework. In: [23], pp. 147-212 (2009)
77. Minsky, M.: "The Society of Mind", Simon and Schuster, New York (1986)

78. Nguyen G., Dang T.T, Hluchy L., Laclavik M., Balogh Z., Budinska I.: AGENT PLATFORM EVALUATION AND COMPARISON, Institute of Informatics, Slovak Akademy of Sciences (2002)
79. Nwana, H., & Wooldridge, M., Software Agent Technologies. BT Technology Journal 14(4), pp. 68-78. (1996)
80. Nwana, H.S.: "ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems", Proceedings of PAAM'98, London pp. 377-392. (1998)
81. Nwana, H.S., Ndumu, D.T., Lee, L.C., Collis, J.C.: " A Toolkit and Approach for Building Distributed Multi-Agent Systems ", Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, WA, USA, pp. 360-361. (1999)
82. Pokahr, A., Brauhach, L., Lamersdorf, W.: Jadex: A BDI Reasoning Engine. In [20], pp. 149-174, Springer (2005)
83. Rao, A.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language, Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world, Einhoven, Netherlands, pp. 42-55 (1996)
84. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems, MIT Press (2001)
85. Ross, R., Collier, R., O'Hare, G.: AF-APL: Bridging principles and practices in agent oriented languages. In Programming Multi-Agent Systems, Second Int. Workshop (ProMAS'04), volume 3346 of LNCS, Springer Verlag, pp. 66–88, (2005)
86. Russell, S., and Norvig, P.: Artifical Intelligence: A Modern Approach. Prentice-Hall, 2nd edition, 2002.
87. Santoro, C.: Towards an Agent Programming Language, In 10th national workshop Towards the Future of Agent-based software systems, Parma, Italy (2009)
88. Sardina, S., Lespérance, Y.: Golog Speaks the BDI Language. In: 7th International Workshop on Programming Multi-Agent Systems, ProMAS 2009, LNCS 5919, pp. 82-99, Springer (2010)
89. Schiffel, S., Thielscher, M., Trang, D.T.: An Agent Team Based on FLUX for the ProMAS Contest 2007, Proceedings of the 5th international conference on Programming multi-agent systems, ProMAS'07, LNCS 4908, pp. 261-265, Springer-Verlag (2008)
90. Shoham, Y.: "Agent-Oriented Programming", Artificial Intelligence, 60(1), pp. 51-92 (1993)
91. Suna, A., Seghrouchni, A.E.F.: Programming mobile intelligent agents: An operational semantics, Web Intelligence and Agent Systems, vol.5, no.1, pp. 47-67, IOS Press (2007)
92. Sycara, K.P., Paolucci, M., Velsen, M.V., Giampapa, J.A.: The RETSINA MAS Infrastructure, Autonomous Agents and Multi-Agent Systems, Springer, no.1-2, pp. 29-48 (2003)
93. Thielscher, M.: Introduction to the fluent calculus. Electronic Transactions on Artificial Intelligence, 2(3–4), pp. 179–192 (1998)
94. Thielscher, M.: Reasoning Robots. The Art and Science of Programming Robotic Agents, Applied Logic Series, Vol.33, Springer (2005)
95. Thomas, R.S.: "PLACA, an Agent Oriented Programming Language", PhD thesis, Computer Science Department, Stanford University, Stanford, CA 94305, (1993)
96. Unland, R., Klusch, M., Calisti, M.: Software Agent-Based Applications, Platforms and Development Kits, Birkhauser Verlag AG (2005)

97. Schip, R.C.v.h., Warnier, M., Brazier, F.M.: Deploying BDI agents in open, insecure environments, in: Proceedings of the 7th European Workshop on Multi-Agent Systems (EUMAS'09) (2009)
98. Wagner, G.: VIVA knowledge-based agent programming. Preprint, Institut fur Informatik, Universitat Leipzig, Germany, (1996)
99. Wang, J.B., Pang, J., Jiang, B.C.: The Modeling and Implementation of Virtual Enterprise Based on Multi-Agent System, Applied Mechanics and Materials (Volume 33) pp. 280-284. (2010)
100. Weerasooriya, D., Rao, A. S., Ramamohanarao, K.: Design of a Concurrent Agent-Oriented Language, LNAI, Vol 890, Springer-Verlag, (1994)
101. Weiss, G. (ed.): Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. MIT Press 2000.
102. Weiss, G.: Agent orientation in software engineering. Knowledge Engineering Review, 16(4), pp. 349–373. (2002)
103. Winikoff, M.: JACKTM Intelligent Agents: An Industrial Strength Platform, In: [20], Springer, pp. 175-193, (2005)
104. Wooldridge, M., Jennings, N.R.: "Agent Theories, Architectures, and Languages: A Survey", Intelligent Agents, LNAI, Vol 890, Springer- Verlag, pp. 1-39. (1994)
105. Wooldridge, M., Jennings, N.R.: "Intelligent Agents: Theory and Practice", available as http://www.doc.mmu.ac.uk:80/STAFF/mike/ker95/ker95-html.html, (1994)
106. Wooldridge, M.: A Knowledge-Theoretic Semantics for Concurrent MetateM, In Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, LNCS, Vol. 1193, pp. 357 – 374, (1996)
107. Wooldridge, M.: Intelligent Agents, chapter 1 of Multiagent systems: a modern approach to distributed artificial intelligence, Massachusetts Institute of Technology, 2000, pp. 27 – 77 (2000)
108. Jinkai, X., Weihong, Y.: Study of comparison between JAFMA and JADE, Circuits, Communications and System (PACCS), 2010 Second Pacific-Asia Conference on, pp. 105 – 108 (2010)
109. Brooks, R.A.: Elephants don't play chess, In Robotics and Autonomous Systems, vol 6., pp 3-15, (1990)
110. Lee, E.A.: Cyber Physical Systems: Design Challenges, EECS Department, University of California, (2008)
111. http://www.cs.uu.nl/3apl/, accessed in January 2011
112. http://www.eil.utoronto.ca/aac/abs/, accessed in January 2011
113. http://www.agentfactory.com/index.php/AF-AgentSpeak, accessed in Jan. 2011
114. http://www.agentscape.org/, accessed in January 2011
115. http://www.cougaar.org/, accessed in January 2011
116. http://fipa.org/, accessed April 2011
117. FIPA-OS (http://fipa-os.sourceforge.net/index.htm) accessed in January 2011
118. http://www.cs.toronto.edu/cogrobo/main/systems/index.html, accessed in January 2011
119. http://jade.tilab.com/, accessed in January 2011
120. http://jadex-agents.informatik.uni-hamburg.de/, accessed in January 2011
121. http://jason.sourceforge.net, accessed in January 2011
122. http://www.madkit.org/, accessed in January 2011
123. http://www.csc.liv.ac.uk/~anthony/metatem.html, accessed in January 2011
124. [http://www.ai.sri.com/~oaa/], accessed in January 2011
125. http://www-2.cs.cmu.edu/~softagents/, accessed in January 2011

126.http://www.ai.mit.edu/people/sodabot/slideshow/total/p001.html, accessed in January 2011
127.http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm, accessed in January 2011
128.http://agents.media.mit.edu/, accessed in January 2011
129.http://193.113.209.147/projects/agents/zeus/index.htm, accessed in January 2011
130.http://www.aosgrp.com, accessed in January 2011
131.http://www.i-a-i.com/, accessed in January 2011
132.http://mmi.tudelft.nl/~koen/goal.php, accessed in January 2011
133.http://apapl.sourceforge.net/, accessed in January 2011

Costin Bădică received in 2006 the title of Professor of Computer Science from University of Craiova, Romania. He is currently with the Department of Software Engineering, Faculty of Automatics, Computers and Electronics of the University of Craiova, Romania. His research interests are at the intersection of Artificial Intelligence, Distributed Systems and Software Engineering. He authored and coauthored more than 100 publications related to these topics as journal articles, book chapters and conference papers. He prepared special journal issues and coedited 4 books in Springer's Studies in Computational Intelligence series. He coinitiated the Intelligent Distributed Computing -- IDC series of international conferences that is being held yearly. He is member of the editorial board of 4 international journals. He also served as programme committee member of many international conferences.

Zoran Budimac holds position of full professor since 2004 at Faculty of Sciences, University of Novi Sad, Serbia. Currently, he is head of Computing laboratory. His fields of research interests involve: Educational Technologies, Agents and WFMS, Case-Based Reasoning, Programming Languages. He was principal investigator of more then 20 projects and is author of 13 textbooks and more then 220 research papers most of which are published in international journals and international conferences. He is/was a member of Program Committees of more then 60 international Conferences and is member of Editorial Board of Computer Science and Information Systems Journal.

Hans-Dieter Burkhard is Senior professor at the Institute of Informatics at Humboldt University of Berlin. He founded the Artificial Intelligence group at Humboldt University. He has studied Mathematics in Jena and Berlin, and he has worked on Automata Theory, Petri Nets, Distributed Systems, VLSI Diagnosis and Knowledge Based Systems. Current interests include Cognitive Robotics, Distributed AI, Agent Oriented Techniques, Machine Learning, Socionics, and AI applications in Medicine. He is a fellow of the ECCAI, and he was Vice President of the International RoboCup Federation. His publication activities include numerous papers and book articles, invited talks and memberships in program committees. His soccer robot teams have won several first places in the RoboCup world championships.

Mirjana Ivanović holds position of full professor since 2002 at Faculty of Sciences, University of Novi Sad, Serbia. She is head of Chair of Computer Science. She is author or co-author of 13 textbooks and of more then 230 research papers on multi-agent systems, e-learning and web-based learning, software engineering education, intelligent techniques (CBR, data and web mining), most of which are published in international journals and international conferences. She is/was a member of Program Committees of more then 80 international Conferences and is Editor-in-Chief of Computer Science and Information Systems Journal.

Received: February 14, 2011; Accepted: April 12, 2011.

Table 3. Summary of agent languages

Name	Web page	IDE	Implementation language	Agent platform integration	Applications	Paradigm	Text book
AGENT0	No	No	Interpreters written in Prolog and CommonLisp	N/A	N/A	Declarative	No
PLACA	No	No	None (experimental)	N/A	N/A	Declarative, prototype	No
Agent-K	No	No	Prolog	N/A	N/A	Declarative	No
MetateM	No	No	Interpreters written in Prolog and Scheme	N/A	According to [Fisher 1994], can be used in process control, fault-tolerance, bidding, etc.	Declarative, based on discrete, linear temporal logic	No
APRIL	http://sourceforge.net/projects/networkagent/	Yes	C, Java, Prolog	No, although its execution relies on external software (e.g. April Machine, InterAgent Communication server)	Networked intelligent agents (kaccording to [McCabe 1994])	Process-oriented symbolic language, not designed specifically for multi-agent programming	No
MAIL	No	No	APRIL	N/A	N/A (development of the language was discontinued)	Hybrid	No
VIVA	No	No	PVM-Prolog	N/A	N/A	Declarative, combining concepts of Prolog and SQL	No
GO!	http://sourceforge.net/projects/networkagent/	Yes	C, Java, Prolog	N/A	Networked intelligent agents	Hybrid (according to [Bordini 2006])	No
Agent Speak	No	Yes, (indirectly (e.g. for Jason)	Several interpreters for the language exist, such as Jason, SIM_Talk, and AgentTalk	N/A	N/A	Declarative, theoretical language	Yes [22]
Jason	http://jason.sf.net	Yes	Java interpreter for AgentSpeak(L)	Yes, based on JADE and Saci; was also integrated in AgentScape [97] and Agent Factory [113]	N/A	Hybrid (according to [21])	Yes [22]
AF-APL	http://www.agentfactory.com/index.php/Main_Page	Yes, via Agent Factory	Java	Agent Factory	Robotics, virtual and mixed reality environments, and mobile computing [Collier, 2009]	According to [21] is hybrid	No

Name	Web page	IDE	Implementation language	Agent platform integration	Applications	Paradigm	Text book
3APL	http://www.cs.uu.nl/3apl/	Yes	A Java implementation and a Haskell implementation	N/A	Robot control using an API called ARIA (provided by http://www.activmedia.com), look at http://www.cs.uu.nl/3apl/thesis/verbeek/verbeekimpl.html	According to classification of [21] is hybrid	No
2APL	http://apapl.sourceforge.net/	Yes	Java on top of JADE	JADE	N/A	Hybrid	No. There is a tutorial
JACK	http://aosgrp.com/products/jack/index.html	Yes	Java	No, execution relies on the JACK agent kernel runtime	Unmanned Aerial Vehicles, surveillance, air traffic management	Imperative	A number of manuals and tutorial.
JADEX	http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview	Yes	Java	JADE	Workflow execution, self-organizing systems, treatment scheduling for patients in hospitals	Hybrid	A number of user guides and tutorials
GOAL	http://mmi.tudelft.nl/trac/goal	Yes	Java based on SWI-Prolog	According to [62], GOAL has been tested on top of JADE. However, we could not find any reference to such an experiment	It is just a prototype that is currently used for educational purposes. It can be useful in planning applications, e.g. in the transportation domain	Declarative	No. There is a tutorial on its Web site
Golog	http://www.cs.toronto.edu/cogrobo/main/	No	Prolog (Eclipse Prolog, SWI-Prolog)	N/A	Cognitive robotics, embedded systems	Declarative	Yes [84]
FLUX	http://www.fluxagent.org/home.htm	No	Two implementations available: 1. Eclipse Prolog (constraint logic programming system), and 2. Sicstus Prolog	N/A	Cognitive robotics	Declarative (also according to [21])	Yes [Thielscher, 2005a]
CLAIM	?	?	Java	SyMPA	N/A	Although in [41] it is said that CLAIM is declarative our impression is that it is hybrid	No