

Agents, Multi-Agent Systems and Declarative Programming: What, When, Where, Why, Who, How?

Matteo Baldoni¹, Cristina Baroglio¹, Viviana Mascardi²,
Andrea Omicini³, and Paolo Torroni³

¹Dipartimento di Informatica, Università degli Studi di Torino,
c.so Svizzera, 185 - I-10149, Torino, Italy
{baldoni, baroglio}@di.unito.it

²DISI, Università degli Studi di Genova,
Via Dodecaneso 35 - I-16146, Genova, Italy
viviana.mascardi@unige.it

³DEIS, ALMA MATER STUDIORUM–Università di Bologna
V.le Risorgimento, 2 - I-40136, Bologna, Italy
{paolo.torroni, andrea.omicini}@unibo.it

Abstract. This chapter tackles the relation between declarative languages and multi-agent systems by following the dictates of the five Ws (and one H) that characterize investigations. The aim is to present this research field, which has a long-term tradition, and discuss about its future. The first question to answer is “*What? What are declarative agents and multi-agent systems?*”. Therefore, we will introduce the history of declarative agent systems up to the state of the art by answering the question “*When? When did research on them begin?*”. We will, then, move to the question “*Where? Where can it take place?*”: in which kind of real applications and for which kind of problems declarative agents and MAS have already proven useful? Connected to where is “*Why? Why should it happen?*”. We will discuss the benefits of adopting the abstractions offered by declarative approaches for developing communication, interaction, cooperation mechanisms. We will compare with other technologies, mainly service-based and object-oriented ones. “*Who? Who can be involved?*”: in order to exploit this kind of technology what sort of background does a specialist have to acquire? We address this question by looking at the Italian landscape of Computer Science research and education. Finally, with the question “*How? How can it happen?*” we will shortly report some examples of existing declarative languages and frameworks for the specification, verification, implementation and prototyping of agents and MAS.

1 What? Declarative Agent Systems

The notion of *declarative agent system* should be taken as a conventional one, to be used in order to focus on an essential theme in agent-oriented computing, rather than to clearly delimit the boundaries of a well-defined research subfield. In fact, given the ever-lasting relationship between agents and MAS, on the one hand, and declarative approaches, languages and technologies, on the other, declarative agent systems are not so easily distinguishable from the notion of MAS themselves. For instance, by adopting

the *strong* notion of agency as promoted by [181], mentalistic notions, like beliefs and desires, with an obvious declarative taste are at the core of the very notion of agent. More specifically, they are at the core of the notion of intelligent agent.

Of course, when adopting weaker notions of agent, such as weak agents in [181], or the autonomy-grounded definition of agent in the A&A metamodel [133], declarative approaches and techniques are no longer strictly required, at a first glance. Whenever MAS are adopted to build up non-trivial systems, however, declarative technologies are typically the only viable approach, mainly due to the high-level of abstraction over complexity they promote [134].

Declarative Agents. Historically, one landmark of declarative agent system is represented by Shoham's AgentSpeak [156], the pioneering framework for agent-oriented programming (AOP) promoting a mentalistic view of agents based on components such as beliefs, decisions, capabilities, and obligations, and where the mental state of agents is described formally in an extension of standard epistemic logics. According to [182]

AOP may be regarded as a kind of "post-declarative" programming. ... In AOP, the idea is that, as in declarative programming, we state our goals, and let the built-in control mechanism figure out what to do in order to achieve them. In this case, however, the control mechanism implements some model of rational agency ... Hopefully, this computational model corresponds to our own intuitive understanding of (say) beliefs and desires, and so we need no special training to use it. Ideally, as AOP programmers, we need not be too concerned with how an agent achieves its goals.

Even more so, languages ranging from AgentSpeak(L) [141] to Jason [26], based on the AOP framework, clearly show how declarative and procedural techniques cannot be but combined when building intelligent agents. In particular, when declarative knowledge, required to represent the mentalistic structures of an intelligent agent, needs to be properly combined with procedural knowledge so as to result in an effective process of practical reasoning. More practically, Prolog-like syntax and operators for beliefs, rules, goals, and plans in Jason provide a clear example of how declarative and logic-based technologies are the most suitable approach for the engineering of intelligent agents, covering some of the essential intra-agent aspects of MAS.

Declarative Multiagent Systems. Individual aspects, however, are far from covering all the issues of declarative agent systems. In fact, a huge space for declarative and logic-based approaches is represented by agent societies: intuitively, the social level is where the complexity of MAS typically grows [134]. Handling a MAS composed by hundreds or thousands of agents, as an open system where both known and unknown agents coexist and interact in an unpredictable way, is obviously more than a challenge to MAS engineers. For this very reason, the social level is the one where declarative models and technologies are likely to provide the most relevant contribution: for instance, by allowing system properties to be assessed at design time, and then enforced at run time by suitable declarative technologies, independently of the MAS dynamics, and of the MAS environment as well.

Agent communication languages (ACL) have represented the first and most immediate representatives for the use of declarative technologies in the construction of agent

societies. In particular, it is well-known that languages as KQML [108] and FIPA ACL [77] represent the first and the current standards, respectively, for inter-agent communication. However, while it is simple to understand how speech-act communication can be based on a declarative approach, ACL are only the first example of declarative techniques adopted for the construction of agent societies.

Even though it was not meant to address the problem of MAS coordination, one of the milestones of declarative technologies for MAS is represented by the work on Shared Prolog [31], a Linda-based language for the coordination of Prolog agents. There, for the first time, a logic-based language was used to coordinate a number of concurrent agents, thus exploiting a declarative technology in order to govern interaction within an agent society. Subsequently, other declarative and logic-based languages were conceived and developed for the construction of agent societies based on coordination abstractions, such as the coordination language ReSpecT: there, both the messages and the social rules of are specified in terms of first-order logic tuples hosted in distributed logic-based tuple centres [132].

A step further is represented by the notion of social integrity constraints [7], which formalizes social concepts such as violation, fulfilment, social expectation within a logic-based framework, concepts that can be enforced at run-time through a suitably-defined logic-based infrastructure. At the infrastructural level, declarative technologies are essential in the definition of the concept of MAS institution. This is the case of Basic Institutions, formally defined in [39], and founded on the social interpretation of agent communicative acts, and of Logic-based Electronic Institutions [172], first-order logic tools aimed at the specification of open agent organizations.

Declarative Agent Systems

Overall, it is apparent that declarative languages and techniques are essential in the design and development of modern MAS, where they are typically used to address most critical aspects. Both intra- and inter-agent issues, in fact, are more and more faced by adopting declarative approaches, the most relevant of which are presented in the remainder of this chapter. So, in the end, it would be quite artificial to draw a line between declarative and non-declarative agent systems: more easily, it is typically the case that one should devise those portions of (nearly) any MAS that exploit declarative and logic-based technologies.

2 When?

The history of declarative agent systems partially coincides with that of intentional systems in Artificial Intelligence: the notion of an intelligent agent as an entity which appears to be the subject of beliefs, desires, commitments, and other mental attitudes [156] is well known and accepted by many researchers. The philosopher Dennett coined the term *intentional system* to denote systems of this kind [50]. In that period (after STRIPS), Artificial Intelligence posed a great emphasis on the use of formal representations, often associated with deductive forms of reasoning [180], and logic programming developed very fast, producing languages that allow for writing executable specifications [160].

Since intelligent software agents must be programmed using languages that can be compiled or interpreted, as any other piece of software, the need for programming languages that could fill the gap between the logical theory and the practical issues concerned with software agents' development arose very soon. Computational logic emerged as a natural tool for developing approaches and solutions, in regards to many aspects. First of all, for the formalization of state-related information (*knowledge, beliefs, goals, environment*). Moreover, for the formalization of *behavior*, and therefore, of the skill of reasoning in order to find new information, to take decisions, to build plans. Generally, to produce proper reactions to the environment and to other agents.

The first real implementation of an intentional system was SRI's Procedural Reasoning System, PRS [82,97], developed to represent and use an expert's procedural knowledge for accomplishing goals and tasks, based on the research on procedural reasoning carried out at the Artificial Intelligence Center, SRI International. Procedural knowledge amounts to descriptions of collections of structured actions for use in specific situations. PRS supported the definition of real-time, continuously-active, intelligent systems that make use of procedural knowledge, such as diagnostic programs and system controllers. In order to formalize intentional systems, different logics were developed, among which the theory of intentions [38], the Belief-Desire-Intention (BDI) logic [142], and that of Knowledge-Abilities-Results-Opportunity [168]. The success of these first implementations gave new impulse to the use of logic approaches for representing and giving a semantics to agents and to their behaviors. A noteworthy example in this respect is Wooldridge's Ph.D. thesis [178], which paved the way to research on agent theories, architectures and languages [129].

Shoham's paper *Agent-Oriented Programming* [156] describes one of the first attempts to define a programming language based on intentional notions. The mental categories upon which Agent-0 is based are *belief* and *obligation* (or *commitment*). *Decision* (or *choice*) is treated as obligation to oneself. Relevant is also dMARS [54], implemented at the Australian AI Institute under the direction of Georgeff, which was a kind of second generation PRS, implemented in C++ and used for commercial agent development projects [83].

These first attempts bear the ambition of developing an approach that more fully draws from the experience of computational logic. A first proposal in this direction is METATEM [66]. So in the '90s, there was, on the one hand, a strain towards the engineering of agents and agent systems in order to meet the requirements of commercialization. To this respect it is important to mention AAIL, spun out of Agentis International¹ to address the commercialization of the developed technology, and Agent Oriented Software (AOS), formed by a number of ex-AAIL staff to pursue agent technology developing JACK Intelligent Agents [35].

On the other hand, METATEM proved the importance of computational logic for the feasibility of the verification of properties, like interoperability, of complex (agent) systems. New themes started to be tackled. In order to reason about systems of agents it is in fact necessary to represent also the other agents' beliefs and goals, and to represent in a declarative way the rules that govern their interactions and the communication. It is also important to introduce processes of negotiation and to deal with possibly conflict-

¹ <http://www.agentissoftware.com/>

ing sets of goals. This led to the proposal of languages like Golog [114] and ConGolog [48], of approaches for the representation of interaction protocols like those of Singh [157], and proposals like AgentSpeak(L) [141] which aimed to help the understanding of the relation between practical implementations of the BDI architecture such as PRS and the formalization of the ideas behind the BDI architecture using modal logics [143]. It is important to notice how, in the same years, the revamp of programming languages exploiting garbage collection, such as Java, and the greater efficiency of hardware due to the technological advancements brought a renovated attention onto declarative programming languages due to their ability of dealing with the openness, the dynamicity, and the flexibility that characterize complex systems.

3 Where? Applications

The exploitation of declarative agent systems for industrial projects and applications has a long and successful history dating back to the early and mid nineties. In the following, we provide some meaningful examples coming from different application domains—some of which developed and tested in real, safety-critical scenarios. In the overall, they show that an agent-oriented solution adopting declarative techniques can be fruitfully exploited to satisfy concrete industrial needs, and demonstrate as well the success of declarative agent technologies and systems outside the boundaries of academia.

Among the oldest applications of declarative agents we may mention a re-implementation of TEAM-CPS [175] where agents used the PRODIGY planning system [122] for local network planning, and the Agent-Orientated Programming framework for communication and control. In 1997, Leckie et al. [110] developed a prototype agent-based system for performance monitoring and fault diagnosis in a telecommunication network, where agents were implemented using C5 [148], based on the OPS5 rule language [75], and communicated using KQML.

ARCHON (ARCHitecture for Cooperative Heterogeneous ON-line systems [99]) was Europe's largest ever project in the area of Distributed Artificial Intelligence. It was employed for monitoring and controlling the cycle of generating, transporting and distributing electrical energy to industrial and domestic customers, for the Iberdrola company, one of the world's leading private energy groups. ARCHON's Planning and Coordination Module was implemented as a rule-based system.

In [152], Schroeder et al. describe a declarative and reactive diagnostic agent based on extended logic programming. Both the inference engine used for computing diagnoses and the reactive layer that implements a meta-interpreter for the agent were implemented in Prolog extended with communication facilities.

The IMPACT agent framework [12] integrates concepts from deontic logic and was used to develop real applications. They include combat information management where IMPACT was used to provide yellow pages matchmaking services and aerospace applications where IMPACT technology led to the development of a multiagent solution to the "controlled flight into terrain" problem.

Moving to nowadays, [154] describes Space Shuttle Ground Processing with Monitoring Agents. JESS, the Java Expert System Shell [78], is used to realize a system that helps the monitoring of all the processes, instrumentation and data flows of the

Kennedy Space Center's Launch Processing System. The system, called NESTA, helps to monitor and above all to discover problems concerning the "ground process", i.e. the set of operations carried out in the weeks before the Space Shuttle's launch. NESTA autonomously and continuously monitors shuttle telemetry data and automatically alerts NASA shuttle engineers if it discovers predefined situations.

Daimler A.G. is exploiting BDI-agent features to develop a "goal-context" modeling technique for describing and executing agile business processes. The goal-context approach aims at (i) having a modular process model that describes the process' steps separate from the process' goals and contexts; and (ii) having different modeling levels, for the different parts of the process model. The goal-context approach was used for the engineering change management process of Daimler, and Jadex [29] was employed for developing a running prototype [34]. The change management real application is currently being implemented using the Whitestein agent platform [147,33]. Go4Flex² is a follow-up of these activities, where the goal-context approach will be applied to another area at Daimler.

Other applications of Jadex include a prototype developed for a company to use semantic Web Technologies for improving the search [174], and two simulations based on real (company) data. The first one dealt with logistics in a big packet delivery company [146]. In the second scenario Jadex was used to build a patient scheduling system evaluated using statistical data from over 3000 patient cases collected at the Klinikum Kulmbach hospital [186].

In a recent project that involved DISI, the Computer Science Department of Genoa University, and Ansaldo Segnalamento Ferroviario, the Italian leader in design and construction of signalling and automation systems for conventional and high speed railway lines, a MAS prototype was developed which monitors processes running in a railway signalling plant, detects functioning anomalies, and provides support to the early notification of problems [30]. The prototype was implemented and tested using DCaseLP [117]. Due to the intrinsic rule-based nature of monitoring agents, Prolog proved extremely suitable for their implementation.

In the past, DCaseLP and its ancestor, CaseLP, were used for many industrial research projects: the Kicker project, based on a previous "freight train traffic" application [42], was developed within the framework of the EuROPE-TRIS Project as a result of an industrial collaboration with the Information Systems Division of Italian Railways (Ferrovie dello Stato s.p.a.), and dealt with the train dispatching problem. CaseLP was employed for the design and development of a working prototype of a vehicle monitoring system, which was carried out in collaboration with Elsag s.p.a. [11]. Finally, a prototype of a multimedia, multichannel, personalized news provider [49] was developed using CaseLP in collaboration with Ksolutions s.p.a. as part of the ClickWorld project, a national, Ministry-funded research project.

2APL has been employed for virtual training systems in the TNO research organization in the Netherlands. In [88] 2APL is used as an example to illustrate how a virtual training system can be modeled, whereas in [89] some experiments are reported in which 2APL agents are used to generate explanations in virtual training systems.

² <http://jadex.informatik.uni-hamburg.de/go4flex/>

Descriptions of industrial applications of commercial BDI style agent systems include [61] and [21] which cover the JACK and Agentis agent platforms, respectively. Whitestein's Living Systems technology³ has been applied in scenarios spanning from telecommunications to logistics, supply chain management, and manufacturing.

4 Why? Benefits

The reasons of the success of the agents and declarative programming binomial is that declarative approaches are particularly suitable to handle the complexity of agent systems. Agent systems are dynamic, in the sense that at runtime agents can enter and exit the system and they can be modified at any moment. The interacting are heterogeneous, they have their own goals and they may need to define agreements for cooperating. Declarative languages abstract away from the execution mechanisms and, by merging semantics and computation, they allow the study of a solution and of its properties in the world of concepts. Although in the industry there seems to be a minor interest towards declarative languages, many concepts introduced by declarative approaches are adopted by more widely spread languages and tools. *Bytecode, scripting, assertions, pattern matching, destructuring, correctness* are a few examples. Agent and multi-agent systems based on declarative approaches supply very effective mechanisms for communication, interaction, and cooperation that can be used to implement choreographies, interaction protocols and orchestrations.

These are particularly useful in addressing computing problems which share with multi-agent systems the properties of openness, dynamicity, and flexibility, involving a large number of heterogeneous components that are physically distributed and that interoperate. Some examples are Web Services [184,9], Mashups [100], SOA [151], Sensor Networks [106], Middleware [98], Distributed components [138]. These developments require the specification of proper interfaces that make components accessible through standard protocols and make it possible to develop new applications by combining and integrating existing components. To this aim, components should bear some public information about themselves, their structure, the way in which they are supposed to be used, and so forth. This information should be represented according to some conventional formalism which relies on well-founded models, upon which it is possible to define access and usage mechanisms. In the following we briefly highlight some areas in which the declarative approach is clearly emerging as predominant w.r.t. other approaches.

Exemplary is the case of the Semantic Web, where declarative languages are becoming very important in the Semantic Web, and where the focus started to shift from the *ontology layer* to the *logic layer*, with a consequent need of expressing rules and of applying various forms of reasoning [2], an interest also witnessed by the RULE Markup Language initiative, by the creation of a W3C working group to define a Rule Interchange Format [3].

The Internet itself provides interesting hooks to the declarative languages community. For instance, OASIS, with the language BPEL4WS [130] (a de facto standard for

³ <http://www.whitestein.com/autonomic-technology-platform/overview>

the specification of single services, allowing the representation of a local view of the interaction that should take place, i.e. the interaction from the point of view of the process), has emphasized the need of a language that can be used both as an execution language for specifying the actual behavior of a participant in a business interaction, and noticeably as a *modeling* language, for specifying the interaction at an abstract level. The need of an *abstract representation* that can be reasoned about emerged even more notably for the *composition* of services. Here it is crucial to have tools that allow the verification at design time of properties regarding the behavioral aspects of the composed system. Although proposals have been made for composition rules and models, like BPMN [176] and WS-CDL [102], a comprehensive solution is currently lacking (BPEL4WS is not enough) and research is moving towards considering declarative approaches [167,137,123].

For instance, the work by Zaremski and Wing on software components matching, based on a logic representation of their preconditions and effects [185], inspired most of the work on semantic matchmaking for Web Service discovery. Semantic Web approaches commonly describe services in terms of inputs, outputs, preconditions and effects [1,153]. Inputs and outputs are usually expressed by ontological terms, while preconditions and effects are often expressed by means of logic representations. Amongst the works on semantic matchmaking, Paolucci et al. [135] propose four degrees of match (exact, plugin, subsumes, and fail). These matches tackle representations, in which services are described by means of inputs and outputs; specifically, matches are computed on the ontological relations of the outputs of an advertisement for a service and a query. The advantage of these kinds of match is that a service description does not need to exactly correspond to the request: this flexibility fosters the re-use of Web Services. The work by Zaremski and Wing also influenced the Web Service Modeling Ontology proposal [62], an organizational framework for Semantic Web Services.

On the other hand, often services are not sought to be used individually but rather to be used *jointly* for executing tasks that none of them alone can accomplish. Semantic annotations of the kind “inputs, outputs, preconditions, and effects” are not sufficient in this case: it becomes useful to introduce a notion of *goal* [139,171,14], which can be used to guide both the selection and the composition of services. The introduction of goals strengthens the need of adopting declarative agents. Agents, in fact, include the ability of dealing with goals and performing goal-driven forms of reasoning; agents also feature autonomy and proactivity, which help when dealing with open environments, allowing for instance a greater *fault tolerance* and an easy approach to *exception handling* [119,140,28].

Besides being used as modeling languages and for reasoning in a goal-driven manner, declarative approaches are starting to gain attention also as a means for designing *behaviors*, replacing more traditional (in the area of Business Process Management) procedural approaches and languages, like Petri nets [144] and PI-calculus [121]. The reason (see [137]) is that systems which allow users to maneuver within the process model or change it while working are considered as the most suitable for dynamic process management. Traditional approaches, having an imperative nature, appear to be too rigid as they strictly prescribe how to work, often forcing an overspecification, which as a side effect compromises dynamism. Opposed to the imperative approaches, Pesic

and van der Aalst [137] have proposed *ConDec*, a language for modeling and enacting dynamic business processes. A ConDec model mainly consists of a set of activities and a set of relationships that constrain the way activities can be executed, and are referred to as *constraints*. Constraints can be interpreted as *policies/business rules*, and may reflect different kinds of knowledge, e.g., external regulations and norms, internal policies and best practices, service/choreography goals. Differently than in the prescriptive approaches, where what is not explicitly modeled is forbidden, ConDec models are open: activities can be freely executed, unless they are subject to constraints. This choice has two implications. First, a ConDec model accommodates many different possible executions, improving flexibility. Second, the language provides abstractions to explicitly capture not only what is mandatory, but also what is forbidden. In this way, the set of possible executions does not need to be expressed extensionally and models remain compact. Agent research too explored similar approaches to obtain openness, flexibility, and heterogeneity. Yolum and Singh [183] propose to adopt the notion of commitment to provide a declarative semantics to the interaction protocols: an agent (the debtor) makes a commitment to another agent (the creditor) to bring about a certain property. Commitments capture and handle mutual obligations which relate interacting agents, giving a meaning to the exchanged messages in terms of their impact on commitments. The adoption of commitments allows a greater flexibility in two respects: the interacting parties can be heterogeneous in their implementations as long as they have the ability of understanding the social commitments and of reasoning about them; their executions do not have to attain to a rigidly encoded behavior, but just not to violate the commitments. The commitment approach has been studied also by others such as [76,86].

A social approach, closer to Logic Programming, has been developed within the SOCS EU Project (see also Section “How?”) where global interactions protocols are specified by means of the SCIFF language [6] and its Abductive Logic Programming (ALP) semantics [101]. Protocols are specified only by considering the external observable behavior of interacting entities, and by the concept of expectation about desired events and interaction. Events and expectations are linked by way of forward rules. The SCIFF language comes with an associated proof procedure, used to verify at runtime (or a posteriori, by analyzing a log of the interaction) whether interacting agents conform to the interaction protocols defined. The SCIFF approach is starting to attract the attention of researchers working on Web Services because of its potential as a tool for verifying the interoperability and for giving an executable semantics to languages like ConDec [124]. The SCIFF framework has also been used to implement commitments [164] via a reactive version of the event calculus [37]. A discussion of commitments and expectations together is proposed in [166].

Another issue in which agents’ declarative approaches proved their usefulness and that also gained attention in other fields is trust negotiation. Trust negotiation [18,22,93] is an approach to security and privacy preserving interactions in open networked environment. In such scenarios peers often interact without any previous relationship and without sharing any common security domain. As a consequence, traditional authentication is sometimes undesirable and frequently impossible. Access control policies and privacy policies are based on user properties. Such properties can be encoded in various ways, including digital credentials, unsigned declarations, and reputation measures

[23]. Some proposals for declarative languages that allow the representation of different kinds of policies (e.g. XACML [131] and P3P [173]) have been made by standardization committees and for reasoning about them [24].

Bordering with Trust Negotiation is Argumentation theory [58], where logic models for debate and negotiation are used for modeling agent reasoning and dialogue. The possibility of structuring rational discussion aimed at reaching mutually acceptable conclusions, and the potential for intuitive, modular and tractable implementations are promising tools in all those fields where there is the need of testing the validity of certain kinds of evidence.

5 Who? Required Background

In 1987, while GULP was founded and IJCAI was held in Italy, the main Italian ICT and consumer electronics event, SMAU, was just discovering AI [19]. The heterogeneous mix of AI promoters included small enterprises of academic roots, such as Delphi, a University of Pisa's spin-off then based in Viareggio, and big actors such as IBM, and included many more in between. Back then AI mainly meant Expert Systems, and the use of Prolog inference engines and the adoption of declarative technologies in general was considered a very promising approach. Nixdorf Italia, involved in Esprit-2 research projects and in the development of air fleet optimization tools for Alitalia, was using a development environment written in Prolog, called Twice [120]. IBM, Unisys, Pirelli Informatica and Datitalia Processing, among others, were all promoting expert systems for configuration and diagnosis which made use of knowledge bases and declarative rules. IBM was pushing expert systems technologies by announcing a series of AI courses.

As discussed in [149] the interest for declarative solutions seemed to fade in the years that preceded 2000. In spite of that, declarative programming started to be taught at Italian universities and a growing number of AI-related courses put a significant emphasis on Prolog and rule-based languages. In 2007, GULP ran a survey to evaluate the extent of computational logic teaching at Italian universities. It turns out that nowadays declarative programming is being taught in 20 Italian universities at around 50 courses, at various levels in computer science and engineering curricula. Some of these courses have been running for as long as 20 years. They are sometimes elective courses attended by small classes. In many cases, however, they are fundamental courses (programming methodologies, AI, logics) attended by large classes with as many as 150 students. In 80% of the cases, the syllabus includes practical lab sessions that teach students how to use SWI Prolog, SICStus, ECLIPSe or other Prolog engines, ASP solvers such as DLV, SAT solvers and model checkers. Every year, around 1500 university students over the country attend on average 20 hours of lectures on computational logic topics, 80% of which focus on logic programming. This is an immense heritage. Many graduates who join the labour market master the basics of declarative languages and technologies.

In more recent times, a number of applications of logic programming have been developed, mainly by academic actors, and most of them were never fully fledged [43]. However, even if the majority of Italian software companies chose not to endorse the

declarative paradigm, most of Italian programmers and software engineers do have the necessary background to start working with rules, knowledge bases and inference engines. The effects of this situation are cultural rather than practical. Declarative technologies do not play a major role in implementing systems, but they nevertheless influence the way many programmers and software engineers conceive the systems they implement. Or, at least, they have the potential to do so.

Agent technologies can bring this potential to the surface and help exploit it. As discussed in Section “What”, declarative agent systems are a collection of paradigms and ways of thinking about software systems, rather than a unique, well-defined engineering solution. Although younger than logic programming as a discipline, autonomous agents and multi-agent systems also started to being taught at Italian universities. These are sometimes a part of software engineering and AI courses, but also live as stand-alone courses.⁴

Who can be involved in declarative agent and MAS technologies then, and what kind of background is required to do so? The answer to the second question is easy: today’s graduates already have such a background, or they can easily acquire it since it is already a part of academic curricula. Then, who can or should be involved?

In our opinion, the ability to think in terms of declarative agent and MAS technologies should be mastered by all software engineers who need to develop systems of some complexity. With a warning. The research effort in this domain is considerable and steady, and we hope to see a constant improvement in the theory and in the tools. However, in approaching the world of agents, today’s software engineers should neither seek for revolutionary solutions nor expect to find out that all they have being using so far has become obsolete. That would be a wrong approach. Instead, they should consider declarative agents as a way of thinking that should guide many separate aspects of a system’s design.

The ideas of goals, capabilities, action, interaction, delegation, commitment, trust, artifact and so on could be exported to so many concrete software engineering problems. This does not necessarily mean that one should use Tropos, Gaia or West2East for requirements elicitation and system design, DCaseLP, Jade, KGP or DyLog for implementing the components, CArAgO for the middleware and SCIFF for monitoring their execution. But we suggest that these be considered as sources of inspiration, because a deep understanding of such technologies will help producing software solutions that are more correct and thus safer, and at the same time more scalable, easier to maintain and monitor, and more suited to integration and interoperation.

6 How? Tools and Languages

In this section we briefly survey (without the presumption of being exhaustive) the tools and the languages that exploit declarative approaches. We structure the presentation in two parts. The former presents the most noticeable BDI-style proposals, while the latter presents approaches based on computational logic. For each of the main proposals, we

⁴ The Universities of Palermo, Genova, L’Aquila, Torino, Bologna, Firenze, Pavia, Roma La Sapienza, Trento, Bari, Modena e Reggio Emilia, Pisa, Milano’s Politecnico and many other ones offer such courses.

describe the same four facets, so to facilitate a comparison: *Semantics*, *Implementation*, *Extensions*, and *Purpose of use*. The last part of the presentation is mainly devoted to Italian research.

6.1 BDI-style tools and languages

AgentSpeak(L) [141] takes as its starting point PRS and its dMARS implementation. It is based on a restricted first-order language with events and actions. Beliefs, desires and intentions of the agent are not represented as modal formulas, but they are ascribed to agents, in an implicit way, at design time. The current state of the agent can be viewed as its current belief base; states that the agent wants to bring about can be viewed as desires; and the adoption of programs to satisfy such stimuli can be viewed as intentions.

Semantics: At run-time, an agent consists of a set of beliefs, a set of plans, a set of intentions, a set of events, a set of actions, and a set of selection functions. The operational semantics is driven by the rules for selecting plans, adopting them as intentions, and executing the adopted intentions [55].

Implementation: There are many implementations of the AgentSpeak(L) language, among which: (a) SIM_Speak [115] (the first AgentSpeak(L) interpreter), which runs on Sloman's SIM_AGENT toolkit, a testbed for cognitively rich agent architectures [158]; (b) Jason [27] that implements, in Java, the operational semantics of an extended version of AgentSpeak(L) (<http://jason.sourceforge.net>).

Extensions: The community working on AgentSpeak(L) is, and has been in the past, very active. Many extensions exist, among which: cooperation through plan exchange [10]; ontological reasoning [126]; belief revision [8]; team formation [96]; combination with the Semantic Web [107].

Purpose of use: The main application of AgentSpeak(L) is in formal verification. Bordini et al. [25] developed model-checking techniques that apply directly to multi-agent programs written in AgentSpeak(L). AgentSpeak(L) multi-agent systems are translated into either Promela or Java models, and then, respectively, SPIN or JPF are used as model checkers.

3APL – “An Abstract Agent Programming Language” [90] – supports the design and construction of intelligent agents for the development of complex systems through the concepts beliefs and procedural goals (also often termed plans). In turn, these can be used to describe and understand the computational system in a natural way. Beliefs represent the issues the agent must deal with, while goals allow the agent both to focus on what it must achieve and to represent the way in which it can achieve it. The practical reasoning rules provide the agent with planning capabilities to find an appropriate plan to achieve a goal, capabilities to create new goals to deal with a particular situation, and capabilities to use the rules to revise a plan.

Semantics: 3APL semantics was originally specified by means of Plotkin-style transition semantics [91] and has been re-specified in Z later on [53]. In [45], the specification of a programming language for implementing the deliberation cycle of cognitive agents is shown, and 3APL has been used as the object language.

Implementation: Both a Java version and an Haskell version of 3APL can be downloaded from <http://www.cs.uu.nl/3apl/>. More recently, a simplified version has been implemented in the Maude term rewriting language [170].

Extensions: The newest incarnation of 3APL is 2APL (A Practical Agent Programming Language) [44]. It can be downloaded from <http://www.cs.uu.nl/2apl/>.

Purpose of use: The 2APL platform which provides a set of tools designed to support the implementation, execution, and testing of multi-agent systems. Its application in the field of virtual training has been discussed in Section 3.

Among the other proposals, it is worthwhile to mention Agent-0 by Shoham [156], which exploits a declarative approach and is the first proposal of an agent-oriented approach to programming. For Shoham, a complete AOP system will include three primary components: (a) A restricted formal language with clear syntax and semantics for describing mental states, the mental state will be defined uniquely by several modalities, such as belief and commitments; (b) An interpreted programming language in which to define and program agents, with primitive commands such as *REQUEST* and *INFORM*; (c) An “agentification process” to treat existing hardware devices or software applications like agents. Agent-0 is targeted towards the second component. Two prototype interpreters were developed: one implemented in Common Lisp, and another developed by Hewlett Packard as part of a joint project to incorporate AOP in the New WaveTM architecture. Agent-0 has two extensions, PLACA [162] and Agent-K [46].

Another interesting tool is Jadex [29], which brings together BDI-style reasoning and FIPA-compliant communication [64] and extends the traditional BDI-model (e.g. with explicit goals). Jadex agents have beliefs, goals, that are implicit or explicit descriptions of states to be achieved, and plans. The Jadex research project is conducted by the Distributed Systems and Information Systems Group at the University of Hamburg. The developed software framework is available under GNU's LGPL license⁵. It allows for programming intelligent software agents in XML and Java and can be deployed on different kinds of middleware such as JADE, a software framework implemented in Java that facilitates development of interoperable intelligent multi-agent systems and that is distributed under an Open Source License [20].

Finally, Dribble [169] is a propositional language that constitutes a synthesis between the declarative features of the language GOAL [92], and the procedural features of 3APL. GOAL agents do not provide planning features, but they do offer the possibility to use declarative goals to select actions. The language Dribble thus incorporates beliefs and goals as well as planning features. Also worthwhile to mention MYWORLD [179], in which agents are directly programmed in terms of beliefs and intentions; ViP [105], a visual programming language for plan execution systems with a formal semantics based upon an agent process algebra; CAN [177], a conceptual notation for agents with procedural and declarative goals; NUIIN [52], a Java framework for building BDI agents, with strong emphasis on Semantic Web aspects; SPARK [127], that builds on PRS and supports the construction of large-scale, practical agent systems; and JAM [95] that combines ideas drawn from the BDI theories, the PRS system and its UMPRS and PRS-CL implementations, the SRI International's ACT plan interlingua [128], and

⁵ <http://sourceforge.net/projects/jadex/>

the Structured Circuit Semantics (SCS) representation [111]. It also addresses mobility aspects from Agent Tcl [87], Agents for Remote Action (ARA) [136], Aglets [109] and others. A survey of languages for programming BDI-style agents can be found in [116].

6.2 Computational logic-based tools and languages

The IMPACT Agent Language [12] is a relevant example of use of deontic logic to specify agents.

Semantics: The paper [60] provides a series of successively more refined semantics for action programs that compute the set of all action status atoms that are true with respect to an agent program P , the current state S and the set IC of underlying integrity constraints on agent states.

Implementations: The implementation of an IMPACT agent program consists of two major parts, both implemented in Java: (a) the IMPACT Agent Development Environment which is used by the developer to build and compile agents, and (b) the run-time part that allows the agent to autonomously update its “reasonable status set” and execute actions as its state changes.

Extensions: Many extensions to the IMPACT framework are discussed in [161] which analyses meta agent programs to reason about other agents based on the beliefs they hold; temporal agent programs to specify temporal aspects of actions and states; probabilistic agent programs to deal with uncertainty; and secure agent programs to provide agents with security mechanisms. Agents able to recover from an integrity constraints violation and able to continue to process some requests while continuing to recover are discussed in [59]. The integration of planning algorithms in the IMPACT framework is discussed in [56].

Purpose of use: IMPACT’s purpose is to allow the integration of heterogeneous information sources and software packages for solving real problems.

Golog [114] is a logic-programming language based on situation calculus, that allows for reasoning on both atomic and complex actions. ConGolog [48] is the concurrent extension of Golog, and it includes facilities for prioritizing the concurrent execution, interrupting the execution when certain conditions become true, and dealing with exogenous actions. Golog is an alternative to traditional plan synthesis, since it allows forms of procedural planning.

Semantics: The semantics of ConGolog and Golog is based on situation calculus and is in the style of transition semantics.

Implementations: Interpreters have been developed in SWI-Prolog and for ECLIPSE as well (<http://www.cs.toronto.edu/cogrobo/main/systems/>).

Extensions: Many extensions exist: Legolog (*LEGO MINDSTORM in (Con)Golog* [113]), IndiGolog (*Incremental Deterministic (Con)Golog* [47]), CASL (*Cognitive Agent Specification Language* [155]), and an extension of ConGolog with sensing actions [145]. More recently, Golog has been exploited to represent flexible templates of Web service composition and integrate user preferences in the composition process [159]. In [79], the compilation of ConGolog into Basic Action Theories for planning is discussed.

Purpose of use: Golog and ConGolog allow the design of flexible controllers for agents living in complex scenarios. IndiGolog provides a practical framework for real robots that must sense the environment and react to changes occurring in it. CASL is an environment based on ConGolog which provides a verification environment.

Concurrent METATEM [66] is a programming language for distributed artificial intelligence, based on first-order linear temporal logic [65]. A Concurrent METATEM system contains a number of concurrently executing agents which are able to communicate through message passing. Each agent executes a first-order temporal logic specification of its desired behavior.

Semantics: METATEM semantics is the one defined for first-order linear temporal logic.

Implementations: Two implementations have been produced. The first is a prototype interpreter for propositional METATEM implemented in Scheme. A more robust Prolog-based interpreter for a restricted first-order version of METATEM has been used as a transaction programming language for temporal databases [63].

Extensions: Single Concurrent METATEM agents have been extended with deliberation and beliefs [68] and with resource-bounded reasoning [71]. Compilation techniques for MASs specified in Concurrent METATEM are analyzed in [103]. Concurrent METATEM has been proposed as a coordination language in [104]. The definition of groups of agents in Concurrent METATEM is discussed in [69,73]. The research on single Concurrent METATEM agents converged with the research on Concurrent METATEM MASs in the paper [72] where “confidence” is added to both single and multiple agents. The development of teams of agents is discussed in [94].

Purpose of use: In [67] a range of sample applications of Concurrent METATEM utilizing both the core features of the language and some of its extensions are discussed. They include bidding, problem solving, process control, fault tolerance. Concurrent METATEM has the potential of specifying and verifying applications in all of the areas above [74], but it is not suitable for the development of real systems.

SCIFF is a framework, developed within the EU-funded SOCS project,⁶ thought to specify and verify interaction in open agent societies [6].

The SCIFF language is equipped with a semantics based on Abductive Logic Programming (ALP) [101]. Interaction is modeled by way of rules (*Social integrity constraints*), which associate the current state of affairs, including all the relevant events detected so far, with a number of alternative possible future worlds, characterized in terms of what is expected or not expected of them. SCIFF’s operational component is an ALP proof procedure for reasoning with expectations in dynamic environments. The SOCS approach to the specification and verification of agent societies [4], is *open*, aimed at minimally restricting the operation of system components, and it is inspired by the deontic notions of prohibitions and permission.

⁶ Societies Of Computees (SOCS, IST-2001-32530): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. <http://lia.deis.unibo.it/research/SOCS/>

Semantics: The semantics of SCIFF is given as a mapping to ALP, augmented with a notion of consistency of expectations. SCIFF is sound and complete under realistic domain assumptions [6].

Implementations: SCIFF is implemented using Constraint Handling Rules [80]. It runs on SICStus Prolog and on SWI Prolog. SCIFF is also embedded in SOCS-SI, a Java tool for runtime monitoring and verification of agent interaction [5].

Extensions: Recent extensions of SCIFF are an efficient implementation of the Event Calculus for Commitment tracking [165,37], extensions for static verification of declarative models [125], its integration with Tropos [32], its extension for constraint optimization [81], and a number of extensions for several application domains described in the SCIFF⁷ and CLIMB⁸ Web sites.

Purpose of use: SCIFF is used for interaction specification and verification. Its main application domains, beside multi-agent systems, are business processes, Web service choreographies, and medical guidelines.

DCaseLP is a multi-language development environment for Multi-Agent Systems. It provides tools and languages for modelling and implementing a MAS prototype following a set of steps which guide the developer from the late requirement analysis to the prototype implementation. The languages and tools that DCaseLP integrates are UML and an XML-based language for the analysis and design stages, Java, JESS and TuProlog [51] for the implementation stage, and JADE for the execution stage. Software libraries for translating UML class diagrams into code and for integrating JESS and TuProlog agents into the JADE platform are also provided.

Semantics: No unifying formal semantics of the agents and the MAS, despite the language they are modeled or implemented in, have been defined.

Implementations: DCaseLP is implemented on top of JADE and provides libraries for seamless integration of agents implemented in TuProlog or JESS and enriched with FIPA-compliant communication capabilities. It can be downloaded from the web site⁹.

Extensions: A translator from UML sequence diagrams to Prolog agent skeletons that can be embedded into DCaseLP has been developed and integrated within the computer-aided Agent-Oriented Software Engineering West2East framework [36].

Purpose of use: DCaseLP main purpose is fast prototyping of agent systems. Its applications in industrial research projects have been discussed in Section 3.

Dynamics in Logic [17,13] is a programming language for reasoning about actions, that can be used for specifying agents and for executing agent specifications. The authors adopt a modal action theory, in which actions are represented by modalities. The adoption of Dynamic Logic or a modal logic to deal with the problem of reasoning about actions and change is motivated by the fact that modal logic allows a very natural representation of actions as state transitions, through the accessibility relation of Kripke

⁷ <http://lia.deis.unibo.it/research/sciff/>

⁸ <http://lia.deis.unibo.it/research/climb/>

⁹ <http://www.disi.unige.it/person/MascardiV/Software/DCaseLP.html>

structures. Since the intentional notions (or attitudes), which are used to describe agents, are usually represented as modalities, the proposed modal action theory is well suited to incorporate them. The language can represent incomplete belief states and can deal with sensing actions as well as with complex actions.

Semantics. The logical characterization of *Dynamics in Logic* is provided in two steps: (a) a multimodal logic interpretation of a dynamic domain description which describes the monotonic part of the language is introduced; (b) an abductive semantics to account for non-monotonic behavior of the language is provided [17]. The language relies on such abductive semantics to provide a nonmonotonic solution to the frame problem; when there are no ramifications, it has been proved to be equivalent to the language \mathcal{A} .

Implementation. A goal-directed proof procedure for reasoning about complex actions (including sensing actions), which can be considered as an interpreter of the language, is supplied. This procedure can be extended for constructing linear and conditional plans to achieve a given goal from an incompletely specified initial state. The interpreter was implemented in Sicsuts Prolog; it is a straightforward implementation of its operational semantics and is available on request.

Extensions. In [13] the language was extended to represent beliefs of other agents in order to reason about conversations. A communication kit including a primitive set of speech acts, a set of special “get message” actions, was included, allowing for the specification of conversation protocols. Other proposals with a causality operator are presented in [84,85].

Purpose of use. The language *Dynamics on Logic* is suitable for building agents acting, interacting and planning in dynamic environments. A web agent system called *WLog* [15], supplying adaptive services in a web-based application context, has been developed to demonstrate the language potential in developing adaptive web applications as software agents. More recently, the language has been used also for giving a declarative interpretation to web services [14,16].

DALI language and agent architecture [41]. DALI is an agent programming language encompassing basic patterns for reactivity, proactivity, internal thinking, and memory. A DALI agent is a logic program that contains reactive rules, aimed at interacting with an external environment. The reactive and proactive behavior of a DALI agent is triggered by several kinds of events: external, internal, present and past events.

Semantics: The declarative and procedural semantics of DALI is defined as an evolutionary semantics in order to cope with the evolution of an agent corresponding to the perception of events. The semantics has been generalized so as to include the communication architecture by resorting to the general framework RCL (Reflective Computational Logic) based on the concept of reflection principle.

Implementations: The DALI interpreter has been implemented in SICStus Prolog, and includes a FIPA-compliant communication library. The DALI interpreter is in principle able to interoperate with other FIPA-compliant platforms; interoperability with JADE is already guaranteed. DALI agents can be distributed on the web, as the implementation of the communication primitives is based on TCP/IP.

Purpose of use: DALI is suitable to implement reactive agents, embedded in an interactive environment. Cultural heritage applications have been proposed, where DALI

agents discover the users' movements via a Galileo satellite signal and proactively learn and enhance user profiles to competently assist users during their visits [40].

Finally, for more information on computational logics and MAS, we forward the interested reader to a number of comprehensive surveys already available in the literature, among which [150,163,118,70].

Acknowledgments

The authors acknowledge Alexander Pokahr and Lars Braubach for their support in describing Jadex applications, and Maaïke Harbers and Mehdi Dastani for the help in describing 2APL ones. Viviana Mascardi is partially supported by the Iniziativa Software CINI-FinMeccanica Project.

References

1. OWL-S: Semantic markup for web services. <http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/>.
2. Reasoning on the web with rules and semantics, network of excellence. <http://rewise.net>.
3. Rule interchange format. W3C, http://www.w3.org/2005/rules/wiki/RIF_Working_Group.
4. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. The soacs computational logic approach to the specification and verification of agent societies. In *Global Computing, IST/FET International Workshop, GC 2004*, volume 3267 of *LNCIS*, pages 314–339, 2005.
5. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. *Applied Artificial Intelligence*, 20(2-4):133–157, Feb.-Apr. 2006.
6. M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Logic*, 9(4):1–43, 2008.
7. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Modeling interaction using *Social Integrity Constraints*: A resource sharing case study. In Leite et al. [112], pages 243–262. 1st International Workshop (DALI 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.
8. N. Alechina, R. H. Bordini, J. F. Hübner, M. Jago, and B. Logan. Belief revision for AgentSpeak agents. In *Proc. of AAMAS 2006*, pages 1288–1290. ACM, 2006.
9. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.
10. D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini. Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange. In *Proc. of AAMAS'04*, pages 698–705. 2004.
11. E. Appiani, M. Martelli, and V. Mascardi. A multi-agent approach to vehicle monitoring in motorway. Technical report, DISI – Università di Genova, 2000. DISI TR-00-13. Presented at the poster session of the 2nd European Workshop on Advanced Video-based Surveillance Systems, AVBS 2001.
12. K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V. S. Subrahmanian. IMPACT: A platform for collaborating agents. *IEEE Intelligent Systems*, 14(2):64–72, 1999.
13. M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *JLAP, special issue on Web Services and Formal Methods*, 70(1):53–73, January 2007.

14. M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Reasoning on choreographies and capability requirements. *International Journal of BPIM*, 2(4):247–261, 2007.
15. M. Baldoni, C. Baroglio, and V. Patti. Web-based adaptive tutoring: an approach based on logic agents and reasoning about actions. *Artificial Intelligence Review*, 22:3–39, 2004.
16. M. Baldoni, C. Baroglio, V. Patti, and C. Schifanella. Conservative re-use ensuring matches for service selection. In *Proc. of Sixth European Workshop on Multi-Agent Systems, EUMAS 2008*, Bath, UK, December 2008.
17. M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Programming Rational Agents in a Modal Action Logic. *AMAI, Special issue on Logic-Based Agent Implementation*, 41(2-4):207–257, 2004.
18. S. Baselice, P. A. Bonatti, and M. Faella. Policy language specification. Technical Report I2-D2, REWERSE network of excellence, 2007.
19. L. Bazzocchi. Lo SMAU scopre l'intelligenza artificiale. *Office Automation*, pages 86–90, Nov. 1988. Available from <http://www.bazzocchi.com/>.
20. F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
21. S. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In *AAMAS 2006*, pages 10–15. ACM, 2006.
22. P. A. Bonatti, J. L. D. Coi, D. Olmedilla, and L. Sauro. Policy-driven negotiations and explanations: Exploiting logic-programming for trust management, privacy & security. In *ICLP*, volume 5366 of *LNCS*, pages 779–784. Springer, 2008.
23. P. A. Bonatti, C. Duma, N. E. Fuchs, W. Nejdl, D. Olmedilla, J. Peer, and N. Shahmehri. Semantic web policies - a discussion of requirements and research issues. In Y. Sure and J. Domingue, editors, *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, pages 712–724. Springer, 2006.
24. P. A. Bonatti, J. L. D. Coi, , and D. Olmedilla. Protunes technical specifications. Technical Report I2-D12, REWERSE, 2007.
25. R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *JAAMAS*, 12(2):239–256, 2006.
26. R. H. Bordini and J. F. Hübner. BDI agent programming in AgentSpeak using *Jason* (tutorial paper). In F. Toni and P. Torroni, editors, *Computational Logic in Multi-Agent Systems*, volume 3900 of *LNCS*, pages 143–164. Springer, Apr. 2006. Proc. of CLIMA VI.
27. R. H. Bordini, J. F. Hübner, and M. Wooldridge, editors. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.
28. L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. CooWS: Adaptive BDI agents meet service-oriented computing. In *Proc. of WWW/Internet*, pages 205–209, 2005.
29. L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A short overview. In *Main Conference Net.ObjectDays 2004*, pages 195–207, 2004.
30. D. Briola, V. Mascardi, and M. Martelli. Intelligent agents that monitor, diagnose and solve problems: Two success stories of industry-university collaboration. *Journal of Information Assurance and Security*, 4(2):106–116, 2009.
31. A. Brogi and P. Ciancarini. The concurrent language, Shared Prolog. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):99–123, Jan. 1991.
32. V. Bryl, P. Mello, M. Montali, P. Torroni, and N. Zannone. B-tropos: Agent-oriented requirements engineering meets computational logic for declarative business process modelling and verification. In *Proc. of CLIMA VIII*, volume 5056 of *LNCS*, pages 157–176. Springer, 2008.
33. B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa. BDI-agents for agile goal-oriented business processes. In *Proc. of AAMAS 2008*, pages 37–44. IFAAMAS, 2008.

34. B. Burmeister, H.-P. Steiert, T. Bauer, and H. Baumgärtel. Agile processes through goal- and context-oriented business process modeling. In *BPM 2006*, volume 4103 of *LNCS*, pages 217–228. Springer, 2006.
35. P. Busetta, R. Ronnquist, A. Hodgson, and A. Lucas. JACK intelligent agents – components for intelligent agents in Java. *AgentLink News Letter*, 2, 1999.
36. G. Casella and V. Mascardi. West2East: exploiting WEb Service Technologies to Engineer Agent-based SofTware. *IJAOSE*, 1(3/4):396–434, 2007.
37. F. Chesani, P. Mello, M. Montali, and P. Torroni. Commitment tracking via the reactive event calculus. In *Proc. of IJCAI*, pages 91–96, 2009.
38. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42, 1990.
39. M. Colombetti, N. Fornara, and M. Verdicchio. A social approach to communication in multiagent systems. In Leite et al. [112], pages 191–220. 1st International Workshop (DALT 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.
40. S. Costantini, L. Mostarda, A. Tocchio, and P. Tsintza. Dalica: Agent-based ambient intelligence for cultural-heritage scenarios. *IEEE Intelligent Systems*, 23(2):34–41, 2008.
41. S. Costantini and A. Tocchio. The DALI logic programming agent-oriented language. In *Proc. of JELIA*, volume 3229 of *LNCS*, pages 685–688. Springer, 2004.
42. A. Cuppari, P. L. Guida, M. Martelli, V. Mascardi, and F. Zini. An agent-based prototype for freight trains traffic management. In *Proceedings of the FMERail Workshop 5*. Springer-Verlag, 1999.
43. A. Dal Palù and P. Torroni. *25 Years of Applications of Logic Programming*, chapter 11. Volume 6000 of *Dovier and Pontelli [57]*, 2010.
44. M. Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
45. M. Dastani, F. S. de Boer, F. Dignum, and J.-J. C. Meyer. Programming agent deliberation – an approach illustrated using the 3APL language. In *Proc. of AAMAS’03*, 2003.
46. W. H. Davies and P. Edwards. Agent-K: An integration of AOP & KQML. In *Proceedings of the Workshop on Intelligent Information Agents*, 1994.
47. G. De Giacomo, Y. Lespérance, H. Levesque, and S. Sardiña. On the semantics of deliberation in IndiGolog – from theory to implementation. In *Proceedings of KR’02*, pages 603–614. Morgan Kaufmann, 2002.
48. G. De Giacomo, Y. Lespérance, and H. J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.
49. M. Delato, A. Martelli, M. Martelli, V. Mascardi, and A. Verri. A multimedia, multichannel and personalized news provider. In *Proceedings of MIPS 2003*, pages 388–399. Springer-Verlag, 2003. LNCS 2899.
50. D. C. Dennett. *The Intentional Stance*. The MIT Press, 1987.
51. E. Denti, A. Omicini, and A. Ricci. Multi-paradigm Java-Prolog integration in tuProlog. *Sci. Comput. Program.*, 57(2):217–250, 2005.
52. I. Dickinson and M. Wooldridge. Towards practical reasoning agents for the semantic web. In *Proc. of AAMAS’03*, pages 827–834, 2003.
53. M. d’Inverno, K. V. Hindriks, and M. Luck. A formal architecture for the 3APL agent programming language. In *Proc. of ZB’00*, pages 168–187, 2000.
54. M. d’Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In *Proc. of ATAL’97*, pages 155–176, 1997.
55. M. d’Inverno and M. Luck. Engineering AgentSpeak(L): A formal computational model. *Logic and Computation Journal*, 8(3):1–27, 1998.
56. J. Dix, H. Munoz-Avila, and D. Nau. IMPACTing SHOP: Putting an AI planner into a Multi-Agent Environment. *Annals of Mathematics and AI*, 4(37):381–407, 2003.

57. A. Dovier and E. Pontelli, editors. *Twenty-five Years of Logic Programming in Italy*, volume 6000 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010.
58. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
59. T. Eiter, V. Mascardi, and V. S. Subrahmanian. Error-Tolerant Agents. In A. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond – Essays in Honour of Robert A. Kowalski, Part I*, pages 586–625. Springer-Verlag, 2002. LNAI 2407.
60. T. Eiter, V. S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, 1999.
61. R. Evertsz, M. Fletcher, R. Jones, J. Jarvis, J. Brusey, and S. Dance. Implementing industrial multi-agent systems using JACK. In *Proc. of PROMAS 2003*, LNCS, pages 18–48. Springer, 2004.
62. D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres. *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer.
63. M. Finger, P. McBrien, and R. Owens. Databases and executable temporal logic. In Comission of the European Communities, editor, *Proceedings of the Annual ESPRIT Conference 1991*, pages 288–302, 1991.
64. FIPA Home Page. <http://www.fipa.org/>.
65. M. Fisher. A normal form for first-order temporal formulae. In *Proc. of CADE'92*, pages 370–384. Springer-Verlag, 1992. LNCS 607.
66. M. Fisher. Concurrent METATEM – A language for modeling reactive systems. In *Proceedings of PARLE'93*, pages 185–196. Springer-Verlag, 1993. LNCS 694.
67. M. Fisher. A survey of Concurrent METATEM – the language and its applications. In *Proc. of ICTL'94*, pages 480–505. Springer-Verlag, 1994. LNCS 827.
68. M. Fisher. Implementing BDI-like systems by direct execution. In *Proc. of IJCAI'97*, pages 316–321. Morgan Kaufmann, 1997.
69. M. Fisher. Representing abstract agent architectures. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL'98)*, pages 227–241. Springer-Verlag, 1998. LNAI 1555.
70. M. Fisher, R. Bordini, B. Hirsch, and P. Torroni. Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence*, 23(1):61–91, 2007.
71. M. Fisher and C. Ghidini. Programming resource-bounded deliberative agents. In *Proc. of IJCAI'99*, pages 200–205. Morgan Kaufmann, 1999.
72. M. Fisher and C. Ghidini. The ABC of rational agent programming. In *Proc. of AAMAS'02*, pages 849–856. ACM Press, 2002.
73. M. Fisher and T. Kakoudakis. Flexible agent grouping in executable temporal logic. In *Intensional Programming II (ISPLIP'99)*. World Scientific Publishers, 2000.
74. M. Fisher and M. Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Cooperative Information Systems*, 6(1):37–65, 1997.
75. C. Forgy. Ops5 user's manual. Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.
76. N. Fornara and M. Colombetti. A commitment-based approach to agent communication. *Applied Artificial Intelligence*, 18(9-10):853–866, 2004.
77. Foundation for Intelligent Physical Agents (FIPA). *Agent Communication Language Specifications*, 2002.
78. E. Friedman-Hill. *Jess in Action : Java Rule-Based Systems (In Action series)*. Manning Publications, 2002.
79. C. Fritz, J. A. Baier, and S. A. McIlraith. ConGolog, sin trans: Compiling ConGolog into basic action theories for planning and beyond. In *Proc. of 11th Int. Conf. on PKRR*, pages 600–610, 2008.

80. T. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, 37(1-3):95–138, Oct. 1998.
81. M. Gavanelli, M. Alberti, and E. Lamma. Integration of abductive reasoning and constraint optimization in SCIFF. In *ICLP09*, volume 5649 of *LNCS*, pages 387–401. Springer, 2009.
82. M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proc. of AAAI'87*, pages 677–682, 1987.
83. M. P. Georgeff and A. S. Rao. A profile of the Australian AI institute. *IEEE Expert*, 11(6):89–92, 1996.
84. L. Giordano, A. Martelli, and C. Schwind. Ramification and causality in a modal action logic. *Journal of Logic and Computation*, 10(5):626–662, 2000.
85. L. Giordano, A. Martelli, and C. Schwind. Reasoning About Actions in Dynamic Linear Time Temporal Logic. *Journal of the IGPL*, 9(2):298–303, 2001.
86. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic*, 5(2):214–234, June 2007.
87. R. S. Gray, D. Kotz, G. Cybenko, and D. Rus. Agent Tcl. In *Mobile Agents: Explanations and Examples*. Manning Publishing, 1997.
88. M. Harbers, K. van den Bosch, and J. Meyer. Enhancing training by using agents with a theory of mind. In *EduMAS'2009, Proceedings*, pages 23–30. 2009.
89. M. Harbers, K. van den Bosch, and J. Meyer. A study into preferred explanations of virtual agent behavior. In *IVA. 2009*. To appear.
90. K. V. Hindriks, F. S. D. Boer, W. V. der Hoek, and J.-J. C. Meyer. Agent programming in 3APL. *AAMAS Journal*, 2(4):357–401, 1999.
91. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Formal semantics for an abstract agent programming language. In *Proc. of ATAL'97*, pages 215–229, 1997.
92. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. Agent programming with declarative goals. In *Proc. of ATAL'00*, pages 228–243, 2000.
93. T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. Practical declarative network management. In *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 1–10, New York, NY, USA, 2009. ACM.
94. B. Hirsch, M. Fisher, and C. Ghidini. Organising logic-based agents. In *Proceedings of the Second NASA/IEEE Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS II)*, 2002.
95. M. J. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Agents'99, Third International Conference on Autonomous Agents, Proceedings*, pages 236–243, 1999.
96. J. F. Hübner and R. H. Bordini. Developing a team of gold miners using Jason. In *Proc. of ProMAS 2007*, volume 4908 of *LNCS*, pages 241–245. Springer, 2008.
97. F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert Magazine*, 7(6):33–44, 1992.
98. V. Issarny, M. Caporuscio, and N. Georgantas. A perspective on the future of middleware-based software engineering. In *FOSE '07: 2007 Future of Software Engineering*, pages 244–258, Washington, DC, USA, 2007. IEEE Computer Society.
99. N. R. Jennings, E. H. Mamdani, J. M. Corera, I. Laresgoiti, F. Perriollat, P. Skarek, and L. Zsolt Varga. Using Archon to develop real-world DAI applications, part 1. *IEEE Expert*, 11(6):64–70, 1996.
100. A. Jhingran. Enterprise information mashups: integrating information, simply. In *Proc. of VLDB '06*, pages 3–4. VLDB Endowment, 2006.
101. A. C. Kakas, R. Kowalski, and F. Toni. The role of abduction in logic programming. In C. H. D.M. Gabbay and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming 5*, pages 235–324. Oxford University Press.
102. N. Kavantzias, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web services choreography description language version 1.0, 2004. <http://www.w3.org/TR/ws-cdl-10/>.

103. A. Kellett and M. Fisher. Automata representations for concurrent METATEM. In *Proc. of TIME'97*, pages 12–19. IEEE Press, 1997.
104. A. Kellett and M. Fisher. Concurrent METATEM as a coordination language. In *Proc. of COORDINATION'97*, pages 418–421. Springer-Verlag, 1997. LNCS 1282.
105. D. Kinny. ViP: a visual programming language for plan execution systems. In *Proc. of AAMAS'02*, pages 721–728, 2002.
106. D. Klan, K. Hose, and K.-U. Sattler. Developing and deploying sensor network applications with anduin. In *Proc. of DMSN'09*, pages 1–6. ACM, 2009.
107. T. Klapiscak and R. H. Bordini. JASDL: A practical programming approach combining agent and Semantic Web technologies. In *Proc. of DALT 2008*, volume 5397 of *LNAI*, pages 91–110. Springer, 2009.
108. Y. Labrou and T. Finin. Semantics and conversations for an agent communication language. In *Readings in Agents*, pages 235–242. Morgan Kaufmann, 1997.
109. D. Lange and O. Mitsuru. *Programming and Deploying Java Mobile Agents with Aglets*. 1998.
110. C. Leckie, R. Senjen, B. Ward, and M. Zhao. Communication and coordination for intelligent fault diagnosis agents. In *8th IFIP/IEEE International Workshop for Distributed Systems Operations and Management, DSOM'97, Proceedings*, pages 280–291, 1997.
111. J. Lee and E. H. Durfee. Structured circuit semantics for reactive plan execution systems. In *Proc. of AAAI'94*, pages 1232–1237, 1994.
112. J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors. *Declarative Agent Languages and Technologies*, volume 2990 of *LNAI*. Springer-Verlag, May 2004. 1st International Workshop (DALT 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.
113. H. J. Levesque and M. Pagnucco. Legolog: Inexpensive experiments in cognitive robotics. In *Proc. of CogRob2000*, 2000.
114. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.
115. R. Machado and R. H. Bordini. Running AgentSpeak(L) agents on SIM-AGENT. In *Proc. of ATAL'01*, pages 158–174, 2001.
116. V. Mascardi, D. Demergasso, and D. Ancona. Languages for programming BDI-style agents: an overview. In F. Corradini, F. D. Paoli, E. Merelli, and A. Omicini, editors, *WOA 2005: Dagli Oggetti agli Agenti, Proceedings*, pages 9–15. Pitagora Editrice Bologna, 2005.
117. V. Mascardi, M. Martelli, and I. Gungui. DCaseLP: a prototyping environment for multi-language agent systems. In *Proc. of LADS'007*, LNCS, pages 139–155. Springer, 2008.
118. V. Mascardi, M. Martelli, and L. Sterling. Logic-based specification languages for intelligent software agents. *J. of TPLP*, 4(4):429–494, 2004.
119. M. Banzi, G. Caire, and D. Gotta. Wade: A software platform to develop mission critical applications exploiting agents and workflows. In *Proc. of AAMAS 2008*, 2008.
120. W. Mellis. TWAICE: A knowledge engineering tool. *Inf. Syst.*, 15(1):137–150, 1990.
121. R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.
122. S. Minton, C. A. Knoblock, D. R. Kuokka, Y. Gil, R. L. Joseph, and J. G. Carbonell. Prodigy 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, Carnegie-Mellon University, 1989.
123. M. Montali. *Specification and Verification of Open Declarative Interaction Models: a Logic-Based Framework*. PhD thesis, DEIS, University of Bologna, Italy, 2009.
124. M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative specification and verification of service choreographies. *ACM Transactions on the Web*, 2010.

125. M. Montali, P. Torroni, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. Verification from declarative specifications using logic programming. In *Proc. of ICLP 2008*, volume 5366 of *LNCS*, pages 440–454. Springer, 2008.
126. Á. F. Moreira, R. Vieira, R. H. Bordini, and J. F. Hübner. Agent-oriented programming with underlying ontological reasoning. In *Proc. of DALT 2005*, volume 3904 of *LNAI*, pages 155–170. Springer, 2006.
127. D. Morley and K. Myers. The SPARK agent framework. In *Proc. of AAMAS'04*, pages 714–721, 2004.
128. K. L. Myers and D. E. Wilkins. The Act Formalism, Version 2.2. Technical report, SRI International AI Center Technical Report, SRI International, Menlo Park, CA, 1997.
129. H. S. Nwana and D. T. Ndumu. An introduction to agent technology. In H. S. Nwana and N. Azarmi, editors, *Software Agents and Soft Computing*, volume 1198 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 1997.
130. OASIS. Business process execution language for web services v.1.1. 2003.
131. OASIS, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf. *eXtensible Access Markup Language (XACML) Version 2.0*, 2005.
132. A. Omicini and E. Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, Nov. 2001.
133. A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, Dec. 2008. Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.
134. A. Omicini and F. Zambonelli. MAS as complex systems: A view on the role of declarative approaches. In Leite et al. [112], pages 1–17. 1st International Workshop (DALT 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.
135. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proc. of ISWC '02*, pages 333–347. Springer, 2002.
136. H. Peine. ARA - Agents for Remote Action. In *Mobile Agents*. Manning Publishing, 1997.
137. M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006.
138. A. Phung-Khac, A. Beugnard, J.-M. Gilliot, and M.-T. Segarra. Model-driven development of component-based adaptive distributed applications. In *Proc. of SAC '08*, pages 2186–2191. ACM, 2008.
139. M. Pistore, L. Spalazzi, and P. Traverso. A minimalist approach to semantic annotations for web processes compositions. In *Proc. of ESWC*, volume 4011 of *LNCS*, pages 620–634. Springer, 2006.
140. M. Piunti, A. Santi, and A. Ricci. Programming SOA/WS systems with cognitive agents and artifact-based environments. In *Proc. of MALLOW'09 Multi-Agent Logics, Languages, and Organisations Federated Workshops*, CEUR Workshop Proceedings, ISSN 1613-0073.
141. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996. Proc. of MAAMAW'96.
142. A. S. Rao and M. P. Georgeff. Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In *Proc. of IJCAI'91*, pages 498–504, 1991.
143. A. S. Rao and M. P. Georgeff. Decision procedures for BDI logics. *J. Log. Comput.*, 8(3):293–342, 1998.
144. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*. Springer, 1998. the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996.

145. R. Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic (TOCL)*, 2(4):433–457, 2001.
146. W. Renz. Models and multi-agent simulations of logistics networks - a case-study in self-organization by microeconomics. In *MKWI 2008*. GITO-Verlag, Berlin, 2008.
147. G. Rimassa and B. Burmeister. Achieving business process agility in engineering change management with agent technology. In *WOA 2007*, pages 1–7. Seneca Edizioni Torino, 2007.
148. J. Roland, G. Vesonder, and J. Wilson. C5 user manual, release 2.1. Technical report, AT&T Bell Laboratories, 1990.
149. G. Rossi. *Logic Programming in Italy: A Historical Perspective*, chapter 1. Volume 6000 of Dovier and Pontelli [57], 2010.
150. F. Sadri and F. Toni. Computational Logic and Multi-Agent Systems: a Roadmap. Technical report, Department of Computing, Imperial College, London, 1999.
151. J. Salasin and A. M. Madni. Metrics for service-oriented architecture (soa) systems: What developers should know. *J. Integr. Des. Process Sci.*, 11(2):55–71, 2007.
152. M. Schroeder, I. de Almeida Móra, and L. M. Pereira. A deliberative and reactive diagnosis agent based on logic programming. In *Proc. of ATAL'97*, volume 1365 of *LNCS*, pages 293–307. Springer, 1998.
153. Semantic Annotations for WSDL Working Group. Semantic annotations for wsdl and xml schema. Technical report, W3C, 2007.
154. G. S. Semmel, S. R. Davis, K. W. Leucht, D. A. Rowe, K. E. Smith, and L. Boloni. Space shuttle ground processing with monitoring agents. *IEEE Intelligent Systems*, 21(1):68–73, 2006.
155. S. Shapiro, Y. Lespérance, and H. J. Levesque. The cognitive agent specification language and verification environment for multiagent systems. In *Proc. of AAMAS'02*, pages 19–26. ACM Press, 2002.
156. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, Mar. 1993.
157. M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.
158. A. Sloman and R. Poli. SIM_AGENT: A toolkit for exploring agent design. In *Proc. of ATAL'95*, pages 392–407. Springer-Verlag, 1995.
159. S. Sohrabi, N. Prokoshyna, and S. A. McIlraith. Web service composition via the customization of Golog programs with user preferences. In A. T. Borgida, V. K. Chaudhri, P. Giorgini, and E. S. Yu, editors, *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*, pages 319–334. Springer-Verlag, 2009.
160. L. Sterling and E. Shapiro. *The art of Prolog : advanced programming techniques*. 1986.
161. V. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, and R. Ross. *Heterogenous Active Agents*. The MIT Press, 2000. 580 pages.
162. S. R. Thomas. The PLACA agent programming language. In M. Wooldridge and N. R. Jennings, editors, *Proceedings of the 1st International Workshop on Agent Theories, Architectures, and Languages (ATAL'94)*, pages 355–370. Springer-Verlag, 1995. LNCS 890.
163. P. Torroni. Computational logic in multi-agent systems: Recent advances and future directions. *Ann. Math. Artif. Intell.*, 42(1-3):293–305, 2004.
164. P. Torroni, F. Chesani, P. Mello, and M. Montali. Social commitments in time: Satisfied or compensated. In M. Baldoni, J. Bentahar, M. B. van Riemsdijk, and J. Lloyd, editors, *DALT*, volume 5948 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2009.
165. P. Torroni, F. Chesani, P. Mello, and M. Montali. Social commitments in time: satisfied or compensated. In M. Baldoni, J. Bentahar, J. Lloyd, and M. B. van Riemsdijk, editors, *Proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies (DALT-2009)*, Lecture Notes in Computer Science. Springer, 2010.

166. P. Torroni, P. Yolum, M. P. Singh, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. Modelling interactions via commitments and expectations. In *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 263–284, Hershey, Pennsylvania, Mar. 2009. IGI Global.
167. W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohed. Life after BPEL? In *Proc. of WS-FM'05*, volume 3670 of *LNCS*, pages 35–50. Springer, 2005. Invited speaker.
168. B. Van Linder. *Modal Logics for Rational Agents*. PhD thesis, Universiteit Utrecht, Utrecht, The Netherlands, 1987.
169. B. van Riemsdijk, W. van der Hoek, and J.-J. C. Meyer. Agent programming in Dribble: from beliefs to goals using plans. In *Proc. of AAMAS'03*, pages 393–400, 2003.
170. M. B. van Riemsdijk, F. S. de Boer, M. Dastani, and J.-J. C. Meyer. Prototyping 3APL in the Maude term rewriting language. In *Proc. of CLIMA VII*, volume 4371 of *LNCS*, pages 95–114. Springer, 2007.
171. M. B. van Riemsdijk and M. Wirsing. Goal-Oriented and Procedural Service Orchestration. A Formal Comparison. In *AWESOME07*. Durham, UK, September 2007.
172. W. W. Vasconcelos. Logic-visserbased electronic institutions. In Leite et al. [112], pages 221–242. 1st International Workshop (DALI 2003), Melbourne, Australia, 15 July 2003. Revised Selected and Invited Papers.
173. W3C, <http://www.w3.org/TR/P3P/>. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*, 2002.
174. N. Weber, L. Braubach, A. Pokahr, and W. Lamersdorf. Agent-based semantic search at motoso.de. In *MATES 2009*, 2009.
175. T. Weihmayer and M. Tan. Modeling cooperative agents for customer network control using planning and agent-oriented programming. In *IEEE Global Telecommunications Conference, Globecom'92, Proceedings*, pages 537–543. IEEE, 1992.
176. S. White. Business Process Modeling Notation Specification 1.0. Technical report, OMG, 2006.
177. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & procedural goals in intelligent agent systems. In *Proc. of KR'02*, pages 470–481, 2002.
178. M. Wooldridge. *The Logical Model of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK, 1992.
179. M. Wooldridge. This is MYWORLD: The logic of an agent-oriented testbed for DAI. In *Proc. of ECAI ATAL Workshop*, pages 160–178, 1994.
180. M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2002.
181. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, June 1995.
182. M. J. Wooldridge. Issues in agent-based software engineering. In *Cooperative Information Agents*, volume 1202 of *LNCS*, pages 1–18. Springer, 1997.
183. P. Yolum and M. P. Singh. Flexible protocol specification and execution: applying event calculus planning using commitments. In *AAMAS*, pages 527–534. ACM, 2002.
184. Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed. Deploying and managing web services: issues, solutions, and directions. *The VLDB Journal*, 17(3):537–572, 2008.
185. A. M. Zaremski and J. M. Wing. Specification matching of software components. *ACM Transactions on SEM*, 6(4):333–369, 1997.
186. A. Zöllner, L. Braubach, A. Pokahr, F. Rothlauf, T. O. Paulussen, W. Lamersdorf, and A. Heinzl. Evaluation of a multi-agent system for hospital patient scheduling. *International Transactions on Systems Science and Applications*, 1(4):375–380, 2006.