Rule-based Distributed and Agent Systems

Costin Bădică[1], Lars Braubach[2], and Adrian Paschke[3]

[1] Software Engineering Department
Faculty of Automatics, Computers and Electronics, University of Craiova
costin.badica@software.ucv.ro
[2] Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
braubach@informatik.uni-hamburg.de
[3] Corporate Semantic Web
Computer Science Department, FU Berlin
paschke@inf.fu-berlin.de

Abstract. The paper contains an overview of the roles played by rules and rule-based systems in distributed and multi-agent systems. These roles include an overview of traditional and newly emerging application areas as well as internal agent architectures and frameworks implementing these architectures.

1 Introduction

This paper strives to shed light on the connections of rule based systems and multi-agent systems. In order to give an overview about the different areas rules play an important role in, it is coarsely distinguished between application areas of agents with rules and multi-agent system construction aspects. Interesting application areas of rules and agents that will be discussed include rather traditional fields like parallel and distributed rule-based systems, service oriented architecture, grid and peer-to-peer computing as well as upcoming new trends such as cloud computing, rule based wireless networks and complex event processing scenarios. From a construction perspective it will be shown which role rules play in the context of individual agent architectures and also with respect to multi-agents systems as a whole. The former will delve into various rule inspired agent architectures, distinguishing between reactive, deliberative and hybrid approaches while the latter will primarily deal with rules as part of communication, negotiations and also teamwork approaches.

The rest of the paper is structured as follows. In the next Section 2.1 application areas of rules and agents are presented. Thereafter, in Section 3 the construction-related issues of rules in agents and multi-agent systems are discussed. A conclusion is given in Section 4.

2 Rule-Based Distributed Systems

The early relative success of rule-based expert systems employing more efficient rule-based inference engines, pushed forward the application of rule technologies to distributed computing and multi-agent systems. This research direction followed the generalization of rule inference to parallel, as well as to distributed computational models. At least two trends can be observed here: (i) improvement of inference algorithms for rule systems using parallel and distributed systems' technology; (ii) exploiting the more declarative nature of rule-based languages as compared to the procedural languages, for the development of more complex systems composed of autonomous components known as *software agents*.

2.1 Parallel Rule-based Systems

Parallel Forward Chaining Production Systems Advances of computational models for rule-based production systems, mainly related to the development of the RETE algorithm [39] and its extensions for efficient matching of rule patterns and working memory elements [2], but also addressing concurrent processing and activation of rules in production systems, opened a vivid research path, starting in the second half of the '80s and lasting also during the '90s. The main outcome of these researches was the development of powerful implementation technologies for rule-based systems. Note that most of these works use the terms *production* and *production system* as synonymous to rule and rule-based system, so these terms will be interchangeably used in this paper.

Authors of [46] proposed a new parallel architecture for exploiting fine-grained parallelism of forward chaining inference algorithms for rule-based production systems on multiprocessor systems. The main outcome of this work was the significant improvement of the execution speed of a rule-based production system expressed as number of rule firings/second, as well as working memory element changes/second. Their approach targeted all the phases of the forward chaining inference cycle: matching, conflict resolution and right hand side rule evaluation.

Authors of [53], [54] propose an in-depth analysis of concurrent computational approaches for improving the performance of single and multiple rule systems. The authors start with some considerations regarding the performance of rule-based systems. The book covers: (i) *parallel production systems* including algorithms for parallel rule firings, (ii) *distributed production systems* under distributed control, and *multiagent production systems* as well as their related control issues.

Authors of [5] proposed a parallel and distributed version of the RETE algorithm that uses the master-slave paradigm. The pattern matching system is decomposed into master and slave modules, working in parallel. Each component holds a copy of the RETE network. Rules are activated in parallel by master components. When a rule is activated, it sends all the activating facts to an available slave component that performs the activation and returns the results. Therefore rules can be activated in parallel, while computation of the activations is distributed among the slave components.

Note that parallel firing of multiple rules for improving the execution performance of forward chaining production systems can compromise the consistency or the working memory by possible interference of rules' actions and conditions. Solutions to this problem are outlined in [53] and later in [75].

Authors of [3] introduced an architecture that allows parallel production firing by allowing the concurrent execution of the activities of matching, selecting and acting of productions. The architecture is proved correct with respect to the principle of serialization that relaxes the commutativity principle that was proposed by [53].

Parallel Backward Chaining Rule-Based Systems Efficient processing in rule-based systems was also addressed for top down inference engines. An early work is [48] that introduced Backpac, a backward-chained inferencing system designed to run on parallel processor machines. More significant achievements in this area are however related to the development of parallelized versions of the well-known logic programming language Prolog that traditionally uses backward chaining as implementation technology. See [47] for an overview of techniques for parallelizing Prolog programs.

Parallel Deductive Databases Rules played an important role in the development of new databases models, including deductive and active databases [93]. Deductive databases are a suitable model for building large knowledge bases by exploiting both database and knowledge technologies. For example, chapter 6 of [93] contains an overview of parallel processing of rules in production systems, deductive and active databases, while chapter 7 introduces the authors' parallel object-oriented knowledge-based system called $PRACTIC^{KB}$.

Techniques for data and rule partitioning for parallelizing deductive databases are also reported in [94] and [98]. More important, these results were later on exploited for implementing large scale rule reasoning on computer clusters (see Subsection 2.4).

Distributed Jess Our literature review also revealed efforts for distributing classical state-of-the-art rule based systems shells, including Jess[4] [43].

In paper [27] the authors introduce a model for distributing rule-based inference systems called Web of Inference Systems (WoIS). Each member of WoIS is composed of an inference system (IS) and a rule base, while all ISs operate on a single Shared Working Memory (SWM). WoIS is controlled by a dedicated component called manager (M). Each IS holds a copy of a part of the SWM in its local working memory, while all ISs run independently in parallel. This model was utilized to implement a distributed version of Jess called DJess. Synchronization between interfering rules is achieved by means of shadow facts and ghost facts. A *shadow fact* is a Jess fact linked to a Java bean object. Each shared fact is implemented as a shadow fact, and thus an associated Java bean object is created. All the proxies corresponding to the same shared fact are linked together by means of a Java remote object called *ghost fact*. Access of the ISs to

[4] `http://www.jessrules.com/`

the ghost facts are synchronized by acquiring locks during the transition from the conflict resolution stage to the act stage of an inference cycle.

A different approach for distributing Jess called Octopus was reported in the paper [76]. With the Octopus approach several independent Jess engines are interconnected in a star topology as clients of a central server. The server allows them to asynchronously exchange messages using Jess functions for socket communications. The Octopus approach was experimented on a computer cluster running Condor workload management system [90].

2.2 Rule-Based Systems as Agent Reasoning Models

Early works proposed the use of rule-based systems as the basic reasoning model of agents that are part of a multi-agent system. Using this approach, each agent of the system incorporates a rule engine and therefore, its behavior is reduced to performing rule based inference. Agent coordination can be achieved either via a shared working memory or by asynchronous message passing.

Multi-Agent Production Systems In paper [37] it is described a multi-agent system called MAGSY where each agent is a rule-based system. Agents are able to communicate asynchronously, as well as they are able to provide services to other agents. MAGSY is in fact a general-purpose multi-agent framework. It has been applied in practice for distributing solving of transportation and logistics problems. Each MAGSY agent is a triple comprising facts, rules, and services. The agent can receive messages from other agents that trigger the update of their facts. An agent can also invoke services provided by other agents. As a side-effect, service execution can change the agent's sets of facts and rules. Each MAGSY agent performs rule-based inference using a forward-chaining rule-interpreter based on the well-known RETE algorithm [39].

Author of [53] and [54] consider multi-agent production systems, that are conceptually different from parallel and distributed production systems. While parallel production systems emphasize parallel rule matching and firing and distributed production systems emphasize dynamic distribution of productions among the agents of an organization with the goal of execution performance improvement (the response time), multi-agent production systems concern the integration of multiple independent production systems acting on a shared working memory that is useful for their coordination.

Multi-Agent Jess Integration of Jess engine into JADE agents [9] is discussed in [29]. This paper is in fact a tutorial showing how a JADE agent can incorporate a Jess engine with the following functionalities: (i) allowing Jess to capture messages received by the agent as Jess facts; (ii) allowing the agent to send messages to other agents directly from Jess; (iii) implementing the agent behavior as Jess inference. Using this approach it is possible to implement rule-based agents in Jess that interact by exchanging FIPA ACL messages using the JADE middleware (see also rule interaction agents, an example of reactive agent architectures discussed in subsection 3.1).

2.3 Rules for Service Oriented Architecture

2.4 Rule-based Grid/Cloud/High-Performance Computing Systems

A recent research trend can be observed in investigating synergies between high-performance computing and rule-based systems and reasoning. On one hand, the higher expressivity of rule-based languages determines an increase of the computational complexity of the inference algorithms, thus limiting the potential of rule-based systems in applications that require large scale reasoning, as it is for example the Semantic Web [10]. On the other hand, the higher expressivity of rule-based languages can help to improve resource and job management in high-performance computing systems, and thus have the potential for improving the overall performance of these systems. Both trends are briefly reviewed in this section of the paper.

Scalable Rule Reasoning Availability of high-performance computing opened new possibilities for scalable rule reasoning in distributed systems. High-performance computing systems include supercomputers, computer clusters, as well as Grid and more recently Cloud computing infrastructures.

Paper [88] is probably the first reporting the exploitation of the results earlier obtained in parallelizing of deductive databases [94,98] as well as the availability of clusters for parallel computing to investigate the improvement of the reasoning performance for the Semantic Web. The authors of [88] proposed a data partitioning scheme, a parallel algorithm, as well as several optimizations for scalable parallel inference with materialized OWL knowledge bases. The implementation of the algorithm was based on the Jena[5] open source rule-based reasoner and it was experimented on a 16 node computer cluster.

Paper [70] describes MARVIN – a parallel and distributed platform for processing large amounts of RDF data, on a network of loosely coupled peers using a new strategy called divide-conquer-swap. The idea of this approach is to continuously partition the set of RDF triples, compute the closure of each partition in parallel and then swap partitions by exchanges between peers. This technique is shown to eventually reach completeness of reasoning and an efficient strategy called SpeedDate for exchanging data between peers is proposed.

Map-Reduce is a technique for programming large data processing tasks on large computer clusters [33]. Hadoop[6] is an Apache project that "develops open-source software for reliable, scalable, distributed computing" and that also provides a Map-Reduce programming framework. [92] shows how to apply MapReduce on Hadoop for large-scale RDFS reasoning. This work is closely related to: (i) the Large Knowledge Collider (LarKC) project[7] [49] for reasoning with billions of facts and rules that are distributed across different locations, as well

[5] `http://jena.sourceforge.net/`
[6] `http://hadoop.apache.org/`
[7] `http://www.larkc.eu/`

as to (ii) WebPIE[8] [91] – a parallel reasoner based on Map-Reduce which aims at reasoning on the scale of the Web.

Rule-based Workflow and Resource Management for Grid and Cloud Computing Grid and Cloud are modern forms of distributed computing that put a high emphasis on virtualization and software services technologies. Grid is a "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations" [40]. Cloud allows provisioning and utilization of computing power with minimal management effort and minimal knowledge of the infrastructure supporting it. This section briefly presents the role that rules and rule reasoning can play to improve resource and workflow management in the Grid. Most of these results apply also to Cloud computing environments.

Paper [85] introduces a new mechanism for on-demand synthesis of available activities in the Grid by applying ontology rules. Rule-based synthesis combines multiple primitive activities to form new compound activities.

Paper [50] introduces WS-CAM – a rule-based application for collaborative awareness management in grid environments. The idea of this work is to represent complex requirements imposed on Grid environments, either behavioral or functional, as business rules implemented using Drools[9].

Paper [66] presents the Active Grid Information Server providing versatile resource management in grid environments, including resource discovery and selection. The server is using an Event-Condition-Action rule-based system that supports dynamically adjustable schedulers.

A recent trend is the representation of grid scheduling algorithms using rule-based formalisms ([71]). Paper [44] proposes a new rule-based languages called SiLK (Simple Language for worKflows) that provides a rule-chaining representation of scientific workflows. SiLK rule-based workflows can be executed and monitored using OSyRIS (Orchestration System using a Rule based Inference Solution) inference engine, as well as with its distributed version D-OSyRIS. The implementation of OSyRIS is based on Drools. SiLK allowed the rule-based representation of several well-known grid scheduling heuristics.

Flexibility of grid resource management can be enhanced by endowing the Grid with semantic descriptions of resources covering the various software and hardware characteristics, as well as their utilization policies. Grid schedulers can thus benefit of these representations by enhancing monitoring and discovery systems with semantic matchmaking capabilities. Performance of resource discovery can be further improved by exploitation of rule-based systems. For an overview of ontology-based semantic approaches for grid resource management the reader is invited to consult reference [4].

Finally, rule-based approaches were shown to be useful for implementing flexible control strategies and decisions that allow the Grid to achieve Quality of Service commitments required by various applications using a Service Level Agreement (SLA) management system. For example, the authors of the paper

[8] `http://www.few.vu.nl/~jui200/webpie.html`
[9] `http://www.jboss.org/drools`

[72] propose predictive decision rules for adaptive SLA management on the Grid. In [74] a declarative Rule Based Service Level Agreement (RBSLA) framework is described.

2.5 Rule-based P2P Systems

Peer-to-peer (P2P) is a model of distributed systems in which distributed, equally weighted and directly connected peers collaborate by providing resources and services to each other. P2P systems have important applications in distributed processing, distributed content management, and ubiquitous computing. The combination of the decentralization of P2P approach with the declarativeness and flexibility of rules enables the development of new types of intelligent distributed systems. Applications are presented in the domains of heterogenous schema mapping and ubiquitous computing.

Heterogenous Schema Mapping Paper [34] introduces a method for using inference engines to express and process semantics of digital library resources in heterogenous environments. The approach is applied to define metadata mappings between heterogenous schemas in P2P-based digital libraries. The mappings are defined by extracting facts from the XML metadata of resources and then by applying rule-based inference to automatically derive relations between local schemas and other retrieved schemas.

Paper [63] introduces LogicPeer, a P2P extension of Prolog. LogicPeer is a straightforward extension of Prolog with operators that enable goal evaluation over peers in a P2P system. LogicPeer defines two network models: (i) *opaque peer network model* in which each peer does not know the identifiers of its neighbors and a certain query propagation protocol is assumed, and (ii) *transparent peer network model* in which is possible for each peer to obtain the identifiers of its neighbors and thus it allows implementation of customized query propagation protocols. Paper [22] discusses an application of LogicPeer for specifying schema mappings and agents' actions in XML-based data integration tasks.

Ubiquitous Computing Paper [45] presents the use of Mandarax[10] and Sens-ation sensor platform for creating the new SensBution infrastructure for ubiquitous computing. SensBution abstracts the access to sensor data using rule-base inference, while the underlying P2P network propagates queries between peers. Each peer incorporates a rule base and uses it and rule inference to answer the queries received from the other peer via JXTA[11] network programming environment.

Papers [12] and [11] propose a distributed reasoning solution that could be used in ambient environments modeled as P2P networks of agent. Each agent has a partial view of the environment and it holds a locally consistent theory. Local theories are connected via bridging rules, which may result in inconsistency of the global knowledge base. Dealing with the inconsistency is achieved by representing bridging rules with defeasible logics.

[10] `http://mandarax.sourceforge.net/`

[11] `http://jxta.kenai.com/`

Paper [14] introduces the concept of Intelligent Domotic Environment (IDE) that is capable of providing Ambient Intelligence (AmI) to home environments through rule-based reasoning. Firstly, IDE proposes a formalization of the home environment as DogOnt ontology [13]. Secondly, IDE proposes a new middleware called Domotic OSGi Gateway (DOG) based on OSGi[12] that supports interoperability of hardware and software components of the home automation system. Thirdly, IDE properties are defined from the perspectives of what information is necessary (state and structural), as well as of the type of inference required (direct, recursive and multi-stage) for their derivation.

2.6 Rule-based Event Processing Agent Systems

(Complex) Event processing (CEP) is a set of techniques and technologies that helps to understand and control event-driven systems. CEP has emerged as a substantial new field of software engineering and computer science over the last ten years from various research fields addressing event processing. In general, CEP aims at achieving actionable, situational knowledge from distributed systems in real-time or quasi-real-time. CEP tools detect complex event patterns (a.k.a. complex event types) and situations (complex events + conditional contexts), i.e. detecting transitions in the universe of interest that requires action either "reactive" or "proactive" in realtime. It is now one of the fastest growing segments in enterprise middleware software. The decoupled event processing model in distributed event processing systems and in particular intelligent complex event processing systems which exploit rules for processing event messages and making decisions on detected relevant situations can be implemented as event processing networks (EPNs) with distributed event processing agents (EPAs). The Event Processing Technical Society (EPTS) defines an Event processing agent (EPA) (event processing component, event mediator) as a software module that processes events.

Various agent-oriented event processing systems have been developed such as Starview[13], Amit [1], AgentLogic RulePoint [14], Spade [65] and Prova [15].

Amit The core the Amit (Active Middleware Technology) framework is the IBM Situation Manager Rule Language (SMRL) [1] which is a markup language for describing situations, which are semantic concepts in the customers domain of discourse and syntactically equivalent to (complex) event patterns. Events in SMRL have a flat structure, and have a unique name and attributes that can be standard or user defined. The conceptual model defines an event type generalization hierarchy. Amit rule engines are deployed as event processing agents in the active middleware.

[12] `http://www.osgi.org`
[13] `http://www.starviewtechnology.com`
[14] `http://www.agentlogic.com/`
[15] `http://prova.ws/`

RulePoint RulePoint is a server based Event Processing platform based on a reactive agent model. It supports detecting events and is able to responde in reactive manner using event action rule definitions. The agents act as (distributed) realtime alerting systems.

Prova see 3.2

System-S Spade System-S Stream Processing Application Declarative Engine (SPADE) [65]. System S is a large-scale, distributed data stream processing middleware developed at the IBM T. J. Watson Research Center. Its runtime can execute a large number of long-running jobs (queries) that take the form of Data-Flow Graphs. A data-flow graph consists of a set of Processing Elements (PEs) connected by streams, where each stream carries a series of Stream Data Objects (SDOs). The PEs implement data stream analytics and are basic execution units that are distributed over the compute event processing agent nodes. The PEs communicate with each other via their input and output ports, connected by streams.

Starview Remote Agents Starview Remote Agents are based on built-in CEP engines for real-time event processing. The agents can collaborate and cooperate across multiple streams of data by exchanging event messages. The agent follow the "actors" approach where event-processing agents listen for incoming events, and can take action according to predetermined rules.

3 Roles of Rules in Multi-Agent Systems

In this section different roles of rules in agent systems will be presented. This will be done on the micro as well as on the macro layer. The first refers to the meaning of roles for internal agent behavior control, whereas the latter considers rules with multiple agents especially in the context of rule-based interactions.

3.1 Rules on the Micro Layer

The role of rules within agents depends crucially on the *internal agent architecture* employed. According to Wooldridge and Jennings [96, p. 23-24] an agent architecture is defined as follows: “[...] It specifies how [...] the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions [...] and future internal state of the agent. [...]”. The definition highlights the architecture’s responsibility of deducing agent actions and future state on basis of environmental percepts and its current knowledge. One main difficulty of agent architectures is that reactive and deliberative behavior have to be balanced so that an agent is capable of realtime responses to environmental changes as well as planned actions leading

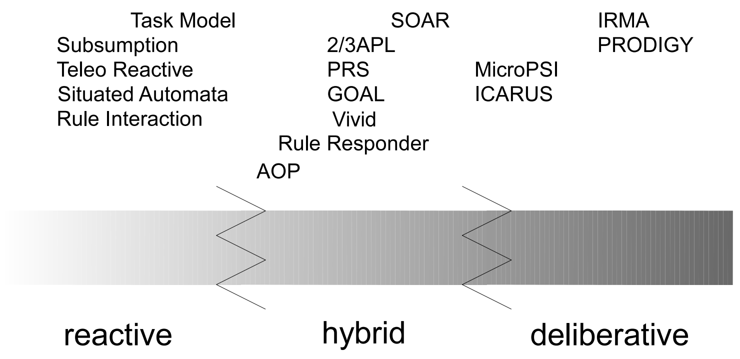

Fig. 1. Agent architecture classification (based on [96])

to achievement of its goals. Reactive capabilities require fast decisions whereas deliberative behavior typically needs time to be thoroughly prepared taking into account possible alternatives and occurring difficulties. The question of how reactive and deliberative behavior should be intertwined is further complicated by the fact that an agent is a resource bounded entity meaning that it has to intentionally devote capabilities to the reactive or deliberative decision making processes. Hence, these limitations led to the development of architectures that disregard one in favor of the other.

These considerations also led to a classification scheme of agent architectures according to the nature of their decision making processes [96]. The class of *reactive agent architectures* emphasizes fast decision making based on sensory input, whereas *deliberative architectures* put the focus on planned actions. *Hybrid architectures* are those that try to combine reactive and deliberative aspects. Figure 1 shows these categorization classes alongside with several agent architectures. The classification scheme is considered here as a spectrum with reactive and deliberative architectures as boundaries and hybrid in the middle. At the left hand side of this spectrum so called purely reactive architectures are located, which represent the event action architectures without a model of the world. On the right hand side the spectrum is bordered by purely deliberative approaches that act purely based on cognitive action often based on planning. The spectrum helps understanding the applicability of different architectures. The idea is not considering one architecture as generally superior to another but understanding the application requirements and matching them to the architecture, i.e. if fast responses are indispensable for an application to function an agent architecture should be located in the reactive or hybrid zone. It has further to be noted that the boundaries between the three categories are rather fuzzy and not all architectures can be clearly assigned to one of the categories. In the following rules are discussed with respect to their role in each of the aforementioned internal agent architecture categories.

Reactive Agent Architectures In a purely reactive rule based agent architecture an agent possesses only reaction rules allowing it to deduce actions from incoming messages or environmental percepts. In the simplest form it can be imagined that the agent behavior completely consists of if-then-else statements containing guards and actions. Its processing would be triggered whenever it receives new input from other agents or the environment. As for nearly all kinds of non-trivial scenarios internal state of an agent is required - otherwise it will repeat wrong behavior over and over again because it cannot remember older outcomes - practical architectures have included mechanisms for autonomous behavior control based on internal state. Though, in contrast to deliberative architectures internal agent state might be kept simple and may not represent a thorough model of the environment. The inclusion of state led to many architectures existing on the boundary between the reactive and hybrid zone and the differences between those and some weakly hybrid architectures are small.

Examples of rather reactive agent architectures include Brook's subsumption architecture [21], task model based architectures [19], as well as several rule based agent architectures (cf. Fig. 1). The subsumption architecture is the prototypical representative for reactive agent architecture and Brooks always insisted on avoiding an internal representation of the world. Despite this fact, even in the subsumption architecture an internal state is preserved in the state machines so that variable values can be saved. Another rather simple but intuitive architecture is the task model, which assumes an agent can be supplied with different behavior snippets called tasks. It has gained some practical attention due to its simplicity and popular agent frameworks such as JADE [8] offering this kind of agent programming abstractions. Both do not rely on rules and thus will not be covered here in more details.

With respect to approaches having relationships to rules teleo-reactive systems [69], situated automata [56] and the rule interaction agent architecture [56] and the rule interaction agent architecture [30] will be discussed.

Teleo-Reactive Agents Nilsson has conceived a reactive agent architecture and programming language with reactive characteristics called teleo-reactive [69]. In order to achieve instantaneous reactions to environmental changes circuit semantics is introduced, i.e. agent actions are not assumed to be executed atomically but need constant conditional support. The architecture assumes that a teleo-reactive agent is constructed from teleo-reactive behaviors, which consist of a set of conditionally guarded atomic actions or subbehaviors. In case the behavior is active all branches of actions are evaluated in parallel. The first branch with a fulfilled condition is then executed until the condition becomes invalid. As subbehaviors can be used as actions, hierarchical execution structures named teleo-reactive trees can emerge at runtime. The architecture shares interesting similarities with production rule systems with some important differences. The guarded actions of a teleo-reactive program could be interpreted as a set of production rules that are evaluated in order to determine the current execution path. The first difference is that production rule systems are typically flat in the sense that all rules are on the same level. In contrast, teleo-reactive programs are hierarchical having earlier layers fulfilling conditions for the execution of deeper layers. Furthermore, production rule systems assume atomic action execution, whereas teleo-reactive actions are executed continuously as long as its guarding condition holds. Agent platforms supporting teleo-reactive agents are AgentMT(TR) [58] and AgentFactory[16] [31].

Situated Automata The situated automaton architecture [56] considers an agent as a finite-state machine whose inputs are fed by environmental sensors and whose outputs are directly connected to its actuators. Considering the agent as finite-state machine expressed as a fixed sequential circuit allows for a efficient execution and thus facilitates reactive responses in dynamic environments. The behavior of a an agent is described using goal reduction rules, which help in mapping higher-level goals into more concrete goals. A compiler is then used to transform the goal rules and top-level goal specification into a simple circuit that is able to map input vectors to output vectors according to the goal rules. This means that symbol manipulation for solving a goal is used only at compile time while at runtime the agent simply behaves according to the generated circuit semantics. To the knowledge of the authors there are no cuurent platforms using the situated automata agent architecture.

Rule Interaction Agents The rule interaction architecture is based on the idea of combining FIPA speech act communication semantics[41,42] with rule oriented behavior descriptions. In general, the agent architecture consists of a rule engine that contains domain behavior rules as well as specific predefined interaction transformation rules. Whenever an agent receives a message with FIPA-SL content it will automatically execute rules that perform a knowledge representation conversion from FIPA-SL to CLIPS assuming specific semantics of SL speech acts, i.e. in case of an 'inform' the receiver will store the new information in its knowledge base, whereas a 'request' performative act directly leads to action execution. For outgoing message the architecture provides a conversion in the opposite direction. The architecture has been realized based for the JESS and Jamocha Rete rule engines [30].

Deliberative Agent Architectures A deliberative agent architecture in its purist sense only consists of a thinking process driving the decisions of the agent. The underlying assumption of deliberative agents is the physical symbol system hypothesis of Newell and Simon [86, p. 35] that states: "A physical symbol system has the necessary and sufficient means for general intelligent action." This strong claim assumes that human thinking is effectively based on symbol manipulation so that machines applying symbol manipulation can act intelligently. Hence, in contrast to a reactive agents the internal representation of the world combined with its processing capabilities enables deliberative agents anticipating incidents and adapt itself accordingly [36]. Typically, an deliberative agent is equipped with achievement goals that describe desired world states and applies problem

[16] `http://www.agentfactory.com/index.php/Main_Page`

solving methods to find a sequence of actions that form a path from its current state to the desired state. Deliberative architectures in many cases use planning, search or rule techniques or a combination of those in order to realize this kind of problem solving. The advanced cognitive capabilities of deliberative agents are often also reflected in additional architecture skills like learning or knowledge deduction.

IRMA (Intelligent Resource-bounded Machine Architecture) [18], MicroPSI [6] PRODIGY [28] and ICARUS [59] represent internal agent architectures with a focus on deliberative processing tasks. IRMA is an architecture, developed by Pollack and Bratman, that tried to directly adopt Bratman's BDI (belief-desire-intention) model of practical reasoning for agent decision making. It uses the aforementioned attitudes belief, desires and intentions and mainly relies on planning techniques to refine partial plans and deduce agent actions. MicroPSI is based on Dörner's PSI (personality-systems-interactions) theory and includes aspects like perception, thinking, emotions, motivation and memory. MicroPSI realizes the PSI theory by relying on a neural network inspired approach. In contrast to these architectures Prodigy and ICARUS, which have been implemented as agent systems, employ some rule based ideas and will be presented in more detail in the following.

PRODIGY The PRODIGY architecture [28] is based on a general problem solver and planner that searches for operator sequences bringing about a set of achievement goals as described in an initial state definition. The search process is guided by control rules that can be domain dependent or independent. Furthermore, PRODIGY enables different kinds of learning mechanisms for control rules. Problem solving in PRODIGY is a two-staged process operating on a tree of nodes, each node representing a world state and the goal set that is to pursue. In the decision phase four kinds of decisions can be controlled via rules: 1) determination of the node to expand, 2) selection of the goal to satisfy, 3) selection of an operator to try, and 4) binding of parameter values of the operator. Thereafter, in the expansion phase the operator is applied and a new node is created for the derived state. In case the operator cannot be executed due to unsatisfied preconditions, a new subgoals for establishing the preconditions are created and also a new node for processing them is created. Control rules in PRODIGY have a specific form. They consist of a left-hand side condition for testing applicability and a right-hand side with an action that can be 'select', 'reject' or 'prefer'. In the first step selection rules are fired to determine the valid set of candidates (node, goal, operator or bindings). If no selection rules trigger all candidates are included. In the next step rejection rules are executed in order to exclude unwanted candidates. Finally, in the last step preference rules are used to order the remaining elements and find the most promising candidates. In case backtracking has to be employed the next most preferred candidate is selected.

ICARUS ICARUS [59] has been conceived as cognitive architecture mainly for controlling agents in complex physical environments. The ICARUS architecture mainly consists of three components: a perceptual, a planning and an execution system. They are meant to operate concurrently and interact using a specific memory system. This memory system is based on a categorization of concepts in tree form. Whenever new percepts are detected by the perceptual module, these experiences are classified in memory using similarity functions of categories. The organization of the memory in form of a lattice and its operation has similarities with a Rete network used for production system matching [38]. At the heart of the architecture the planner module uses means-end analysis to generate plans. It tries to achieve a goal by comparing the goal state with the initial state and then breaking down the problem into subproblems, which are recursively solved by the planner. The planner uses the memory to retrieve suitable operators called skills based on the problems pre- and postconditions or the reduction of differences between the states it can bring about. In case the planner encounters problems it may backtrack resulting in a heuristic depth-first search. The categorized memory allows the planner learn from previous experiences by fetching entire plans that can be used as starting point for the problem at hand and may be subject to further adaptations or refinements. The architecture has been extended with the possibility of specifying a degree of persistency in performing its activities allowing to adjust its degree of reactivity.

3.2 Hybrid agent architectures

Hybrid agent architectures aim at providing a balanced mixture of reactive and deliberative behavior specification means and execution. Due to resource boundedness agent architectures have to solve the question of how much effort to spend for each type of behavior and how often to rethink courses of actions they have committed to. Several experiments have shown the degree of commitment should be dependent on the degree of dynamics exposed by the environment the agent is situated in [78,57]. This has led to the development of architectures with different commitment strategies ranging from bold agents strongly committed to their intentions to cautious agents reconsidering frequently. In contrast to deliberative agent architecture which are often based on planning approaches hybrid architectures rather employ reactive planning or purely rule-based behavior control. Reactive planning, originally stemming from PRS, describes an iterative but very fast planning approach that is based on the idea of planning step by step at runtime taking into account immediate feedback of the environment, i.e. an agent only decides upon the next plan or action and during execution expands subgoals to further plans at runtime. This scheme of acting has been adopted also by many other architectures such as 2/3APL and GOAL, described hereafter.

AOP Agents The AOP (Agent Oriented Programming) architecture [84] envisions a mentalistic agent description based on the notions beliefs, capabilities and commitments. The fundamental idea is that an agent commits to execute an action for another agent or itself at the current or a future point in time. Actions are described as capabilities with a guard determining the applicability in regard to the agent's context. The means for engaging in commitments is based on commitment rules that may include a message as well as a mental condition. In case a commitment rule fires, a new commitment is added and kept until it got executed or belief changes render the capability's condition invalid. If the latter situation occurs the commitment is removed and a notification to the agent the commitment belong to should be prepared. Rule evaluation is done in each agent deliberation cycle. Frameworks using AOP inspired architectures are AgentFactory [31] and AgentBuilder[17] [81].

PRS Agents The PRS (procedural reasoning system) architecture [80] builds on the BDI (belief-desire-intention) model of agency [17], which explains human behavior on basis folk-psychological notions, i.e. the BDI model explains rational behavior in the way humans think that they think. Foundation of the BDI model is the process of practical reasoning, which is composed of two subsequent subprocess: goal deliberation and means-end reasoning [95]. The first refers to the responsibility of deciding what goals to pursue, which might be difficult when goals are conflicting. The latter is concerned with determining on the means how to achieve a previously selected goal. The PRS architecture only considers means-end resoning by casting BDI to beliefs, goals and plans. Goals appear in form of events that trigger a plan selection and execution process (means-end reasoning). Similar to processing event-condition-action rules the PRS interpreter first selects a subset of applicable plans according to the event type and then selects among those using the first plan with fulfilled preconditions. In case of plan errors the means-end reasoning process can be initiated again and other plans may used out until the goal is achieved or the last plan has been chosen. The traditional PRS architecture has also been described as programming language called AgentSpeak(L) [79]. Architectural extensions of PRS have addressed the inclusion of declarative goal semantics by including different goal types like achievement and maintenance [20] as well as conceptual support for the goal deliberation phase [77,67]. Both forms of extensions emphasize the role of rules in the PRS architectures as goal states have to be observed and trigger actions. The PRS architecture has been used in many agent frameworks including JIAC[18], JACK[19], Jason[20] and Jadex[21][15,16].

2/3APL Agents 3APL (an abstract agent programming language) [52] and 2APL (a practical agent programming language) [32] are similar approaches for programming agents using mentalistic notions and rules. As 2APL is the successor of 3APL only the former will be described in the following. A 2APL agent is described by the typical BDI attitudes beliefs, goals, and plans and additionally by three types of rules: planning goal rules, procedure call rules and plan repair rules. Beliefs are specified as Prolog facts or belief inference rules that generate additional knowledge based on the agent's beliefs. Goals are represented as

[17] `http://www.agentbuilder.com/`
[18] `http://www.jiac.de/`
[19] `http://www.aosgrp.com.au/`
[20] `http://jason.sourceforge.net/Jason/Jason.html`
[21] `http://jadex-agents.informatik.uni-hamburg.de/`

formulas describing world states the agent wants to attain. Planning goal rules serve the generation of plans for goals. The condition part of a planning goal rules consist the goal to be present as well as a specific belief state to be valid. The action part contains a plan description composed of an action recipe, which can consist of concrete as well as abstract actions. Procedure call rules are similar to goal planning rules with the difference that as part of the condition instead of goal, message events, environmental events or abstract actions can be used. The usage of both kinds of rules allows for a context based interpretation of goals and runtime expansion of plans. In addition, the third kind of rules called plan repair rule enables reacting on plan failures. Such failures occur when an action of a plan leads to an exception. The condition part of a plan repair rule consists of a belief state check and an action description denoting the beginning of the plan to repair, i.e. the first actions of that plan. The action part contains a replacement plan description that can be used as alternative for the original actions. The 2/3APL architectures have also been implemented in corresponding agent platforms.[22]

GOAL Agents A GOAL agent [51] is a BDI style of agent that makes use of the following types of mentalistic notions. It uses knowledge and beliefs as data structures for storing information. In this respect knowledge represents static facts that will not change during runtime and beliefs contains more volatile data that depends on the perceptual input and received messages. Both kinds of structures may also contain knowledge refinement rules for generating additional deduced facts. The motivations of an agent are synthesized as achievement goals using formulas for describing the desired world states. The program logic is defined by action rules referring to actions that are specified similar to STRIPS actions and make use of pre- and postconditions. An action rule is similar to a production rule consisting of a condition and action part. The condition part is a mental state guard that can e.g. check for goal existence or belief states and the action part contains an action that will be executed when the corresponding rule fires. If more than one action rule is activated a GOAL agent arbitrary selects among them yielding non-deterministic agent behavior. The GOAL architecture is implemented in the GOAL agent system.[23]

SOAR Agents The SOAR (originally for state, operator and result) agent architecture [60,61] has been developed as a candidate for a UTC (unified theories of cognition) [68] helpful for explaining the full gamut of human behavior including e.g. problem solving, learning, and language. SOAR tries to achieve this following the 'parsimony principle', which states that the architecture complexity should be low and it should rely on as few architecture mechanisms as possible. The SOAR architecture is based on the idea of problem solving through operator search and application. In contrast to other architectures SOAR completely relies on production rules for realizing its goal directed behavior. These rules belong

[22] `http://apapl.sourceforge.net/`, `http://www.cs.uu.nl/3apl/`

[23] `http://mmi.tudelft.nl/trac/goal`

to different conceptual groups and their matching and firing is controlled in a sophisticated way by the agent's deliberation cycle. This cycle first transfer sensory input to the SOAR working memory. Thereafter, in the proposal phase, the interpreter fires all activated inference, proposal and comparison rules. Inference rules are used to generate new knowledge from the existing knowledge. Proposal rules serve for operator generation adding them also to the working memory and finally comparison rules are used to establish preferences among the proposed operator instances. In the following decision phase, SOAR has to select exactly one operator. In case the choice is easy and exactly one operator was proposed or one operator is preferred against all others the interpreter directly enters the next application phase. But it may also happen that no operator was selected, several operators are equally well suited or insufficient information is available for operator execution. As the interpreter performs only knowledge decisions it will solve the problem by automatic subgoaling. This means that a subcontext will be established in which SOAR tries to bear new knowledge to resolve the impasse. In the application phase the operator will be executed by firing rules that have been activated by the new operator. In the last phase domain dependent output functions will be called using specific working memory elements as parameters. An implementation of SOAR is developed by an active research community[24].

Vivid Agents Vivid agents [83] is an approach that combines reactive and proactive behavior using rules and planning. The underlying Vivid agent architecture is named CAP (concurrent action and planning) and consists of two independent modules. The reactive module relies on two kinds of rules: reaction and action rules. Reaction rules are similar to event-condition-action rules and are triggered by incoming messages and new environmental percepts. In case the optional condition part is fulfilled the rule is activated and can be fired. In contrast to reaction rules, action rules do not have a triggering event and operate on the agents knowledge only. In Vivid agents three types of (reaction and action) rules are distinguished based on the kind of effects involved: epistemic, physical and communicative. Epistemic rules only have internal effects on the agent's knowledge, whereas physical actions refer to actions performed in the environment and communicative actions deal with sending a message to another agent. The planning module of CAP is responsible for proactive agent behavior. It uses a STRIPS [82] inspired approach that is able to generate a plan for achieving a goal. The planner uses the available action rules as operators and thus produces a plan in form of a sequence of action rules that have to be executed in order to reach the desired world state. After plan generation has been finished the produced plan is executed interleaved with reactive rules. A current implementation of the architecture is not available.

Rule Responder Rule Responder [73] is a Semantic Web infrastructure for distributed rule-based event processing multi-agent eco-systems. The Rule Respon-

[24] `http://sitemaker.umich.edu/soar/home`

der middleware is based on modern enterprise service technologies and Semantic Web technologies for implementing intelligent rule-based agent services that access data and ontologies, receive and detect events (e.g., for complex event processing in event processing agent networks), and make rule-based inferences and (semi-)autonomous pro-active decisions for reactions based on these representations. The core of a Rule Responder agent (cf. Figure 2) are reasoning engines such as the Prova rule engine [25] [26] which implements the decision and behavioral reaction logic of the agents' roles. The Prova rule engine supports different rule types:

– Derivation rules to describe the agent's decision logic
– Integrity rules to describe constraints and potential conflicts
– Normative rules to represent the agent's permissions, prohibitions and obligation policies
– Defeasible rules to priorities rules for, e.g. handling conflicts between agent's goals and modularization of the agent's KB to support multiple roles of an agent
– Reaction rules to define reaction logic which are triggered on the basis of detected (complex) events
– Messaging reaction rules to define the agents conversation-based workflow reactions and behavioral logics based on complex event processing

An agent can employ vocabularies defined as Semantic Web ontologies (e.g., based on RDFS or OWL) or Java class hierarchies to give its rules a domain-specific meaning. The vocabularies can be used within the conversation with other agents to enable a semantic and pragmatic interpretation of the messages, e.g. FIPA ACL pragmatic primitives as semantic ontology concepts in messaging reaction rules.

For the deployment of agents on the Web and for the communication in agent networks, Rule Responder uses an enterprise service bus (ESB) middleware, which supports a multitude of synchronous and asynchronous transport protocols (>40) such as MS, SMTP, JDBC, TCP, HTTP, XMPP, etc. to transport rulebases, queries and answers between the agents. The de facto standard Reaction RuleML [27] is used as a platform-independent rule interchange format for agent conversation using Reaction RuleML messages. Reaction RuleML incorporates various kinds of production, action, reaction, and knowledge representation temporal/event/action logic rules as well as (complex) event/action messages into the native RuleML standard syntax.

Emerald Emerald `http://lpis.csd.auth.gr/systems/emerald/rel.html` like Rule Responder employs reasoning engines as reasoning services that are implemented as reasoner agents, which intercommunicate via FIPA ACL-based

[25] `http://prova.ws`
[26] and other rule engines, such as OO jDREW, DR-Device (initially in Emerald), Euler, or Drools as long as they support Reaction RuleML as general interchange format for agent communication
[27] `http://reaction.ruleml.org`

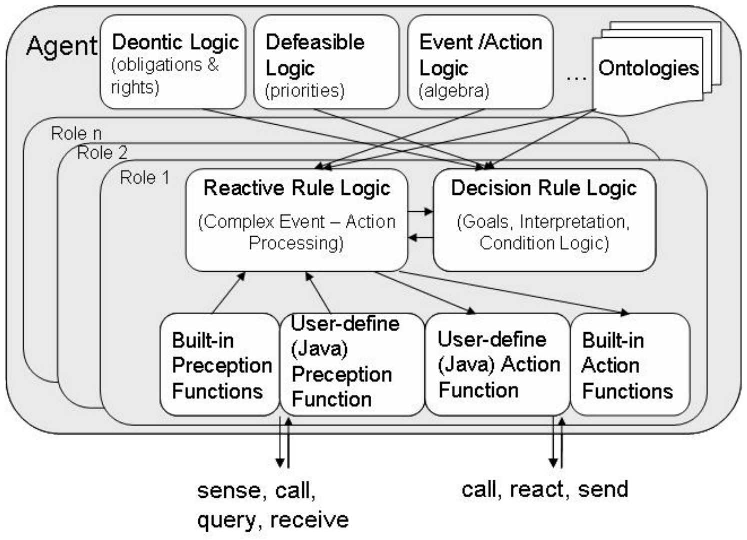
Agent
Deontic Logic
(obligations & rights)
Defeasible Logic
(priorities)
Event /Action Logic
(algebra)
...
Ontologies
Role n
Role 2
Role 1
Reactive Rule Logic
(Complex Event – Action Processing)
Decision Rule Logic
(Goals, Interpretation, Condition Logic)
Built-in Preception Functions
User-define (Java) Preception Function
User-define (Java) Action Function
Built-in Action Functions
sense, call, query, receive
call, react, send

Fig. 2. Rule Responder

communication protocols. EMERALD is built on-top of the JADE multi-agent system. The emerald framework can be used as one platform specific agent framework in the general Rule Responder Semantic Web middleware.

3.3 Rules on the Marco Layer

Rule-Based Negotiation According to [64], *negotiation* is the process by which a group of agents communicate to try to come to a mutually acceptable agreement on some matter. It is one of the important methods for establishing agent cooperation. Understood in this way, negotiation consists of two parts: (i) *negotiation protocol* that represents the conventions under which negotiation operates as a set of public rules of the agents' interaction process. Agents must comply to the protocol of the negotiation in order to be able to communicate; (ii) *negotiation strategy* that represents the specification of the sequence of actions that an agent plans to make during negotiation and that are supposed to lead to a desired outcome.

Negotiation Protocols Rules can be used to define a reusable formalization of the semantics of the interaction between several negotiation participants. The participants are required to obey the rules specific to a given negotiation protocol – for example a certain type of auction. One of the first approaches was proposed by paper [62] that introduces AB3D[28] – a rule-based scripting language for expressing auction mechanisms. AB3D allows initialization of auction parameters,

[28] http://ai.eecs.umich.edu/AB3D/

definition of rules for triggering auction events, declaration of user variables and definition of rules for controlling bid admissibility.

[7] introduces a conceptual framework for the development of agent-based automated negotiations focused on auctions that consists of: (1) negotiation infrastructure, (2) generic negotiation protocol, and (3) taxonomy of declarative rules, is presented. The *negotiation infrastructure* defines roles of negotiation participants and of a host. Participants exchange proposals within a "negotiation locale" managed by the host. The *generic negotiation protocol* defines three phases of a negotiation: admission, exchange of proposals and formation of an agreement, in terms of how, when and what types of messages should be exchanged between the host and negotiation participants. *Negotiation rules* are used for enforcing the negotiation protocol. Rules are organized into a taxonomy: rules for participants admission to negotiations, rules for checking validity of proposals, rules for protocol enforcement, rules for updating the negotiation status and informing participants, rules for agreement formation and rules for controlling the negotiation termination. [25] presents an implementation of the conceptual negotiation framework introduced in [7] in an agent-based e-commerce system. Furthermore, paper [26] presents a representation of negotiation rules using R2ML[29] markup language for English auctions.

[89] proposes a formalization of negotiations that goes beyond the framework of [7]. Its authors suggest usage of an ontology-based approach to expressing negotiation protocols. Specifically, whenever an agent is admitted to negotiation it is to obtain a specification of the negotiation rules in terms of a shared ontology. This approach has been exemplified with a sample scenario by investigating how the ontology can be used to tune the negotiation strategy of participating agents.

Negotiation Strategies In [87] an implementation of a system of Jade agents that negotiate using strategies expressed in defeasible logic was described. The implementation is demonstrated with a bargaining scenario involving one buyer and one seller agent. The buyer strategy was defined by a defeasible logic program.

Rule-Based Verification of Agent Systems and Workflows Rules represented in temporal logics can be used to express patterns of properties verification of concurrent and distributed systems [35], including agent systems and workflows. The pattern approach was taken further in [55] to define a set of verification patterns for checking business process models translated into labeled transition systems, using model checking tools. The pattern approach can considerably simplify the verification process by enabling the reuse of software engineering expertise. This technique was applied for checking an agent-based English auction service [23], as well as different types of middle-agents including frontagents, matchmakers, and brokers [24].

[29] `https://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML`

4 Conclusion

In this paper we have surveyed several major approaches using rules in (multi) agent systems and distributed agent architectures which run rule engines at their core. The approaches differ, e.g., in their supported rule types, state representation, rule evaluation mechanism, conflict resolution and truth maintenance mechanisms. Depending on their expressiveness and semantics the used rule engines might be capable of implementing agents in the strong sense of cognitive architectures for intelligent agents with goal/task-based, utility-based and learning-based functionalities, or in the weak sense of agent services with simple reflexive functionalities for, e.g., deductive query-answering or simple reactive capabilities. Following the general consensus defined by the strong notion of agency in [97], the use of declarative rules for representing the agents' decision and behavioral reaction logics makes them capable of reactive, proactive, and communicative behavior and supports (semi-)autonomous (intelligent) decisions. Additionally, mentalistic notions can be used in the rule language for describing the agent behavior in an abstract and intuitive way, e.g. in the interactions between agents to communicate the pragmatics of the interchanged information.

References

1. Asaf Adi and Opher Etzion. Amit - the situation manager. *VLDB J.*, 13(2):177–203, 2004.
2. José N. Amaral and Joydeep Ghosh. *Speeding Up Production Systems: From Concurrent Matching to Parallel Rule Firing*, chapter 7, pages 139–160. Elsevier, 1994.
3. José N. Amaral and Joydeep Ghosh. A concurrent architecture for serializable production systems. *IEEE Transactions on Parallel and Distributed Systems*, 7(12):1265–1280, 1996.
4. Balachandar R. Amarnath, Thamarai Selvi Somasundaram, Mahendran Ellappan, and Rajkumar Buyya. Ontology-based grid resource management. *Software Practice and Experience*, 39(17):1419–1438, December 2009.
5. Mostafa M. Aref and Mohammed A. Tayyib. Lana-match algorithm: A parallel version of the rete-match algorithm. *Parallel Computing*, 24(5-6):763–775, 1998.
6. J. Bach, C. Bauer, and R. Vuine. Micropsi: Contributions to a broad architecture of cognition. In C. Freksa, M. Kohlhase, and K. Schill, editors, *KI 2006: Advances in Artificial Intelligence, 29th Annual German Conference on AI, KI 2006*, pages 7–18, 2006.
7. Claudio Bartolini, Chris Preist, and Nicholas Jennings. A software framework for automated negotiation. In Ricardo Choren, Alessandro Garcia, Carlos Lucena, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems III*, volume 3390 of *Lecture Notes in Computer Science*, pages 213–235. Springer Berlin / Heidelberg, 2005.
8. F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent systems with JADE*. John Wiley & Sons, 2007.
9. Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons Ltd, 2007.
10. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, (5), May 2001.

11. Antonis Bikakis and Grigoris Antoniou. Distributed defeasible contextual reasoning in ambient computing. In Emile Aarts, James L. Crowley, Boris de Ruyter, Heinz Gerhäuser, Alexander Pflaum, Janina Schmidt, and Reiner Wichert, editors, *Proceedings of the European Conference on Ambient Intelligence, AmI'08*, volume 5355 of *Lecture Notes in Computer Science*, pages 308–325, Berlin, Heidelberg, 2008. Springer-Verlag.
12. Antonis Bikakis and Grigoris Antoniou. Distributed reasoning with conflicts in an ambient peer-to-peer setting. In Max Mühlhauser, Alois Ferscha, and Erwin Aitenbichler, editors, *Constructing Ambient Intelligence*, volume 11 of *Communications in Computer and Information Science*, pages 24–33. Springer Berlin Heidelberg, 2008.
13. Dario Bonino and Fulvio Corno. Dogont – ontology modeling for intelligent domotic environments. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *7th International Semantic Web Conference, ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 790–803. Springer, 2008.
14. Dario Bonino and Fulvio Corno. Rule-based intelligence for domotic environments. *Automation in Construction*, 19(2):183–196, 2010.
15. R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications.* Springer, 2005.
16. R. Bordini, M. Dastani, J. Dix, and A. el Fallah-Seghrouchni, editors. *Multi-Agent Programming: Languages, Tools and Applications.* Springer, Berlin, 2009.
17. M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, 1987.
18. M. Bratman, D. Israel, and M. Pollack. Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence*, 4(4):349–355, 1988.
19. L. Braubach. *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme.* PhD thesis, Universitï¿œt Hamburg, 2007.
20. L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Proceedings of the 2nd International Workshop on Programming Multiagent Systems (ProMAS 2004)*, pages 44–65. Springer, 2005.
21. R. A. Brooks. How to build complete creatures rather than isolated cognitive simulators. *Architectures for Intelligence*, pages 225–239, 1989.
22. Grazyna Brzykcy, Jerzy Bartoszek, and Tadeusz Pankowski. Schema mappings and agents' actions in p2p data integration system. *Journal of Universal Computer Science*, 14(7):1048–1060, 2008.
23. Amelia Bădică and Costin Bădică. Specification and verification of an agent-based auction service. In *Information System Development. Towards a Service Provision Society (ISD 2008)*, pages 239–248. Springer Verlag, 2009.
24. Amelia Bădică and Costin Bădică. Fsp and fltl framework for specification and verification of middle-agents. *Applied Mathematics and Computer Science*, 21(1):9–25, 2011.
25. Costin Bădică, Maria Ganzha, and Marcin Paprzycki. Implementing rule-based automated price negotiation in an agent system. *Journal of Universal Computer Science*, 13(2):244–266, 2007.
26. Costin Bădică, Adrian Giurca, and Gerd Wagner. Using rules and r2ml for modeling negotiation mechanisms in e-commerce agent systems. In Dirk Draheim and Gerald Weber, editors, *Trends in Enterprise Application Architecture, 2nd International Conference, TEAA 2006*, volume 4473 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2007.

27. Federico Cabitza and Bernardo Dal Seno. Djess - a knowledge-sharing middleware to deploy distributed inference systems. In Cemal Ardil, editor, *The Second World Enformatika Conference, WEC'05*, pages 66–69. Enformatika, Çanakkale, Turkey, 2005.
28. J. G. Carbonell, O. Etzioni, Y. Gil, R. Joseph, C. A. Knoblock, S. Minton, and M. M. Veloso. Prodigy: An integrated architecture for planning and learning. *SIGART Bulletin*, 2(4):51–55, 1991.
29. Henrique Lopes Cardoso. Integrating jade and jess. 2007.
30. U. Christoph, K.-H. Krempels, and A. Wilden. Jamochaagent - a rule-based programmable agent. In J. Filipe, A. L. N. Fred, and B. Sharp, editors, *ICAART 2009 - Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 447–454, 2009.
31. R. W. Collier. *Agent Factory: A Framework for the Engineering of Agent-Oriented Applications.* PhD thesis, University College Dublin, 2001.
32. M. Dastani. 2apl: a practical agent programming language. *International Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), Special Issue on Computational Logic-based Agents*, 16(3):214–248.
33. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
34. Hao Ding. Integrating semantic metadata in p2p-based digital libraries. *Library Management*, 26(4/5):218–229, 2005.
35. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc.*21^{st} *international conference on Software engineering (ICSE 1999)*, pages 411–420. IEEE Computer Society Press, 1999.
36. J. Ferber. *Multi-Agents Systems - An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.
37. Klaus Fischer and Hans-Michael Windisch. Magsy: Ein regelbasiertes multiagentensystem. *KI Zeitschrift*, 6(1):22–26, 1992.
38. C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
39. Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
40. Ian T. Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *IJHPCA*, 15(3):200–222, 2001.
41. Foundation for Intelligent Physical Agents (FIPA). *FIPA ACL Message Structure Specification*, December 2002. Document no. FIPA00061.
42. Foundation for Intelligent Physical Agents (FIPA). *FIPA SL Content Language Specification*, December 2002. Document no. FIPA00008.
43. Ernest Friedman-Hill. *Jess in Action: Java Rule-based Systems.* Manning Publications Co., 2003.
44. Marc Eduard Frîncu and Dana Petcu. Osyris: a nature inspired workflow engine for service oriented environments. *Scalable Computing: Practice and Experience*, 11(1):81–97, 2010.
45. Tom Gross, Thilo Paul-Stueve, and Tsvetomira Palakarska. Sensbution: A rule-based peer-to-peer approach for sensor-based infrastructures. In *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 333–340, Washington, DC, USA, 2007. IEEE Computer Society.
46. Anoop Gupta, Charles L. Forgy, and Allen Newell. High-speed implementations of rule-based systems. *ACM Transactions on Computer Systems*, 7(2):119–146, May 1989.

47. Gopal Gupta, Enrico Pontelli, Khayri A. M. Ali, Mats Carlsson, and Manuel V. Hermenegildo. Parallel execution of prolog programs: a survey. *ACM Transactions on Programming Languages and Systems*, 23(4):472–602, 2001.
48. Lawrence O. Hall. Backpac: A parallel goal-driven reasoning system. *Information Sciences*, 62(1-2):169–182, 1992.
49. Frank Harmelen. Large scale reasoning on the semantic web: What to do when success is becoming a problem. In *Proceedings of the 5th International Conference on Active Media Technology, AMT'09*, pages 3–3, Berlin, Heidelberg, 2009. Springer-Verlag.
50. Pilar Herrero, Jose Luis Bosque, Manuel Salvadores, and Maria S. Perez. A rule based resources management for collaborative grid environments. *International Journal of Internet Protocol Technology*, 3:35–45, July 2008.
51. K. Hindriks. Programming Rational Agents in GOAL. In A. El Fallah Seghrouchni, J. Dix, M. Dastani, and R. Bordini, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 119–157. Springer, 2009.
52. K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Meyer. Agent Programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, November 1999.
53. Toru Ishida. *Parallel, Distributed and Multiagent Production Systems*, volume 878 of *Lecture Notes in Computer Science*. Springer, 1994.
54. Toru Ishida. Parallel, distributed and multi-agent production systems - a research foundation for distributed artificial intelligence. In Victor R. Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multiagent Systems, ICMAS*, pages 416–422. The MIT Press, 1995.
55. Wil Janssen, Radu Mateescu, Sjouke Mauw, Peter Fennema, and Petra van der Stappen. Model checking for managers. In *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, volume 1999 of *Lecture Notes in Computer Science*, pages 92–107, London, UK, 1999. Springer-Verlag.
56. L. P. Kaelbling. A situated-automata approach to the design of embedded agents. *SIGART Bulletin*, 2:85–88, July 1991.
57. D. Kinny and M. Georgeff. Commitment and effectiveness of situated agents. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, pages 82–88, February 1991.
58. J. Knottenbelt. *Contract Related Agents*. PhD thesis, Imperial College London, 2006.
59. P. Langley, K. B. McKusick, J. A. Allen, W. Iba, and K. Thompson. A design for the icarus architecture. *SIGART Bulletin*, 2(4):104–109, 1991.
60. J. F. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to Soar, an architecture for human cognition. In S. Sternberg and D. Scarborough, editors, *Invitation to Cognitive Science*, volume 4, pages 212–249. MIT Press, 1996.
61. J. F. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to Soar, an architecture for human cognition. Technical report, University of Michigan, 2006.
62. Kevin M. Lochner and Michael P. Wellman. Rule-based specification of auction mechanisms. In *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 818–825. IEEE Computer Society, 2004.
63. Seng W. Loke. Declarative programming of integrated peer-to-peer and web based systems: the case of prolog. *Journal of Systems and Software*, 79(4):523–536, April 2006.

64. Alessio R. Lomuscio, Michael Wooldridge, and Nicholas R. Jennings. A classification scheme for negotiation in electronic commerce. *Group Decision and Negotiation*, 12(1):31–56, 2003.
65. Bugra Gedik Vibhore Kumar Giuliano Losa Robert Soulé Kun-Lung Wu Martin Hirzel, Henrique Andrade.
66. L. Mohammad Khanli and M. Analoui. Active grid information server for grid computing. *The Journal of Supercomputing*, 50(1):19–35, 2009.
67. V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio. Reasoning about goals in BDI agents: the PRACTIONIST framework. In F. De Paoli, A.Di Stefano, A. Omicini, and C.Santoro, editors, *Proceedings of Joint Workshop "From Objects to Agents"*, 2006.
68. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
69. Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal Artificial Intelligence Research*, 1:139–158, January 1994.
70. Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen. Marvin: Distributed reasoning over large-scale semantic web data. *Journal of Web Semantics*, 7(4):305–316, 2009.
71. Aysan Rasooli Oskooei, Mohammad Mirza-Aghatabar, and Siavash Khorsandi. Introduction of novel rule based algorithms for scheduling in grid computing systems. In *Second Asia International Conference on Modelling and Simulation (AMS 2008)*, pages 138–143. IEEE Computer Society, 2008.
72. James Padgett, Karim Djemame, and Peter Dew. Predictive adaptation for service level agreements on the grid. *International Journal of Simulation: Systems, Science & Technology*, 7(2):29–42, 2006.
73. A. Paschke, H. Boley, A. Kozlenkov, and B. Craig. Rule responder: Ruleml-based agents for distributed collaboration on the pragmatic web. In *Proceedings of the 2nd international conference on Pragmatic web*, ICPW '07, pages 17–28, New York, NY, USA, 2007. ACM.
74. Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated sla management. *Decision Support Systems*, 46(1):187–205, 2008.
75. Tolety Siva Perraju and Bandreddi E. Prasad. Interference analysis in multiple rule firing systems. *Knowledge-Based Systems*, 13(4):171–176, 2000.
76. Dana Petcu and Marius Petcu. Distributed jess on a condor pool. In *Proceedings of the 9th WSEAS International Conference on Computers*, pages 11:1–11:5, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).
77. A. Pokahr, L. Braubach, and W. Lamersdorf. A goal deliberation strategy for bdi agent systems. In T. Eymann, F. Klï¿œgl, W. Lamersdorf, M. Klusch, and M. Huhns, editors, *Proceedings of the 3rd German conference on Multi-Agent System TEchnologieS (MATES-2005)*. Springer, 2005.
78. M. E. Pollack, D. Joslin, A. Nunes, S. Ur, and E. Ephrati. Experimental investigation of an agent commitment strategy. Technical Report 94–31, Department of Computer Science, University of Pittsburgh, 1994.
79. A. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In W. Van de Velde and J. Perram, editors, *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW 1996)*, pages 42–55. Springer, 1996.
80. A. Rao and M. Georgeff. BDI Agents: from theory to practice. In V. Lesser, editor, *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS 1995)*, pages 312–319. MIT Press, 1995.

81. Reticular Systems. *AgentBuilder User's Guide*, version 1.3 edition, 2000. `http://www.agentbuilder.com/`.
82. S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach.* Prentice-Hall, 2003.
83. M. Schroeder and G. Wagner. Vivid agents: Theory, architecture, and applications. *Applied Artificial Intelligence*, 14(7):645–675, 2000.
84. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.
85. Mumtaz Siddiqui, Alex Villazon, and Thomas Fahringer. Semantic-based on-demand synthesis of grid activities for automatic workflow generation. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, E-SCIENCE'07*, pages 43–50, Washington, DC, USA, 2007. IEEE Computer Society.
86. H. A. Simon. Cognitive science: the newest science of the artificial. *Cognitive Science*, 4:33–46, 1980.
87. Thomas Skylogiannis, Grigoris Antoniou, Nick Bassiliades, Guido Governatori, and Antonis Bikakis. Dr-negotiate – a system for automated agent negotiation with defeasible logic-based strategies. *Data & Knowledge Engineering*, 63(2):362–380, 2007.
88. Ramakrishna Soma and Viktor K. Prasanna. Parallel inferencing for owl knowledge bases. In *37th International Conference on Parallel Processing, ICPP'2008*, pages 75–82. IEEE Computer Society, 2008.
89. Valentina Tamma, Steve Phelps, Ian Dickinson, and Michael Wooldridge. Ontologies for supporting negotiation in e-commerce. *Engineering Applications of Artificial Intelligence*, 18(2):223–236, March 2005.
90. Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.
91. Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri E. Bal. Owl reasoning with webpie: Calculating the closure of 100 billion triples. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Part I*, volume 6088 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2010.
92. Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank Harmelen. Scalable distributed reasoning using mapreduce. In *Proceedings of the 8th International Semantic Web Conference, ISWC'09*, volume 5823 of *Lecture Notes in Computer Science*, pages 634–649, Berlin, Heidelberg, 2009. Springer-Verlag.
93. Ioannis Vlahavas and Nick Bassiliades. *Parallel, object-oriented, and active knowledge base systems.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.
94. O. Wolfson and A. Ozeri. Parallel and distributed processing of rules by data-reduction. *IEEE Transactions on Knowledge and Data Engineering*, 5(3):523–530, June 1993.
95. M. Wooldridge. *Reasoning about Rational Agents.* MIT Press, 2000.
96. M. Wooldridge and N. Jennings. Agent theories, architectures, and languages: A survey. In M. Wooldridge and N. Jennings, editors, *Intelligents Agents I, Agent Theories, Architectures, and Languages (ATAL 1994)*, pages 1–39. Springer Verlag, 1995.
97. Michael Wooldrige. *An Introduction to MultiAgent Systems.* John Wiley & Sons, 2002.

98. Weining Zhang, Ke Wang, and Siu-Cheung Chau. Data partition and parallel evaluation of datalog programs. *IEEE Transactions on Knowledge and Data Engineering*, 7(1):163–176, February 1995.

document