Computational Logic and Multi-Agent Systems: a Roadmap

Fariba Sadri, Francesca Toni
Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
{fs,ft}@ic.ac.uk
http://www-lp.doc.ic.ac.uk/~{fs,ft}

December 15, 1999

Abstract

Agent-based computing is an emerging computing paradigm that has proved extremely successful in dealing with a number of problems arising from new technological developments and applications. In this paper we report the role of computational logic in modeling intelligent agents, by analysing existing agent theories, agent-oriented programming languages and applications, as well as identifying challenges and promising directions for future research.

1 Introduction

In the past ten years the field of *agent-based computing* has emerged and greatly expanded, due to new technological developments such as ever faster and cheaper computers, fast and reliable interconnections between them as well as the emergence of the world wide web. These developments have at the same time opened new application areas, such as electronic commerce, and posed new problems, such as that of integrating great quantities of information and building complex software, embedding legacy code. The establishment of the new field of agent-based computing is confirmed by the emergence of a number of regular specialised workshops, conferences and journals, accompanied by the creation of a European Network of Excellence (AgentLink [1]) and of an international organisation for setting standards for the field (FIPA, Foundation for Intelligent Physical Agents [2]).

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

There is no universally accepted definition of agents, but to facilitate our discussion in this paper, and in agreement with most researchers, we take agents to be computing artifacts, hardware or software, that exhibit most of the following properties:

- *Autonomy*, whereby agents have the ability to act without the direct intervention of other agents, including humans.
- *Reactivity*, whereby agents are situated in an environment that they can perceive through sensors and they can act upon in reaction to stimuli. Agents can also revise their beliefs in reaction to new input.
- *Proactivity*, whereby agents are capable of exhibiting intelligent problem-solving behaviour, for example to plan their activities in order to achieve short and long term goals.

A multi-agent system is a collection of agents, each of which, in addition to the above properties, exhibits

- *Social behaviour*, whereby agents are aware of one another, can interact with one another and can modify their behaviour in response to others.

In order to facilitate social behaviour, agents must have some of the following capabilities:

- *Communication*, via a set of (low or high level) constructs and protocols, as well as means of addressing and directing communication.
- *Cooperation*, whereby agents can have joint as well as individual goals. They need the ability to *negotiate* with other agents both to facilitate the achievement of their own goals as well as to construct *joint plans* to achieve joint goals.
- *Belief revision*, in the context of the additional sources of information provided by the other agents.

The area of agent-based computing is orthogonal to a number of traditional computing areas, such as software engineering, distributed computing, human-computer interaction, telecommunication and artificial intelligence.

Within artificial intelligence, a number of approaches have been proposed for representing agents in some logic-based formalism, notably the BDI approach [63] and Agent0 [69]. In the BDI approach an agent is characterised by its *beliefs*, about itself and the other agents with which it interacts as well as the environment in which all agents reside, *desires*, i.e. objectives it aims at, and *intentions*, i.e.

plans it commits to. Beliefs, desires and intentions are represented via modal operators with a possible world semantics. In Agent0 an agent is characterised by its beliefs and its commitments. The commitments are production rules that can refer to non-consecutive states of the world, in which the agent operates. Both the BDI and the Agent0 approaches use logic simply as a tool for representing agents, but rely upon a non-logic-based execution model. This causes a wide gap between theory and practice in these approaches.

In recent years, a number of logic programming-based proposals to agents have been put forward, e.g. [48, 2, 67, 60, 45, 53, 71, 31, 28, 32, 27, 26]. These proposals use (some form of) computational logic both for knowledge representation and as a computational tool (but not necessarily as an implementation tool).

Many past and present developments in computational logic are relevant and sometimes crucial in the modeling and understanding of agents. For example, agents need to reason about changes in the environment, and about how to bring about desired changes. Their performance can be enhanced if they have the ability to learn. They may have to function in an environment about which they do not have complete information, and they may have to choose from a number of alternative courses of actions. They need to modify their beliefs dynamically and to perform integrity checking while doing so. Finally, agents in a multi-agent setting, when cooperating, may need to perform conflict-resolution, possibly by arguing. Thus, developments in computational logic that can make promising contributions include developments in temporal reasoning, defeasible reasoning, abductive reasoning, inductive reasoning, as well as reasoning about actions and planning, reasoning with priorities and preferences, belief revision and argumentation.

A comprehensive survey of agent technology and applications can be found in [43]. This paper complements that survey, in that it reports on approaches to modeling single agents as well as multi-agent systems *in computational logic*, both from a theoretical and applicative perspective. To the best of our knowledge, this report is the first attempt to provide a comprehensive survey of computational logic-based agents and multi-agent systems and to discuss the potential and challenges of this promising and rapidly emerging field.

The report is organised as follows. Section 2 is a survey of existing computational logic-based proposals to agents and multi-agent systems. Section 3 describes some of the research challenges, and section 4 outlines some future opportunities. Section 5 concludes.

2 State of the art

This section is a survey of existing proposals for agent systems based upon computational logic techniques. In this section we will refer to the following formalisms without giving full descriptions: extended logic programming [35], abductive logic programming and integrity constraints [46], and various formalisms for reasoning about actions and time, such as the event calculus [49], the language $\mathcal{A}$ [36], the situation calculus [55] and GOLOG [53]. The event calculus is a logic programming formalism. The original situation calculus is not but can be understood in computational logic terms, and a number of logic programming formalisations of it have been published since [55]. GOLOG is formalised in (second-order) logic and implemented in Prolog.

The remainder of this section is organised as follows. In subsection 2.1 we analyse a number of different approaches from the viewpoint of the knowledge representation their agents rely upon (subsection 2.1.1), the architecture they employ to govern the behaviour of agents (subsection 2.1.2), and the features they exhibit to support multi-agent systems (subsection 2.1.3). In subsection 2.2 we describe the main features of existing languages for programming agents and multi-agent systems. In subsection 2.3 we report on applications and implementations of some existing tools.

2.1 Theory

2.1.1 Knowledge representation

We analyse a number of existing approaches from the view point of their representation of beliefs, desires/goals and intentions. We borrow the ontology of beliefs, desires and intentions from the BDI architecture, without borrowing their specific representation language and semantics. We also analyse the underlying computational mechanisms adopted by the various approaches to obtain reactivity and to perform planning. Planning amounts to proactive behaviour. In general, a plan is a set of (partially or totally) ordered actions and planning is the deliberative process of generating such a set of actions that achieve one or more goals. All the papers we will refer to have given similar logical accounts of planning.

Kowalski and Sadri [48] employ iff-definitions, integrity constraints and observations to represent beliefs of agents. Iff-definitions are extensions of the usual Clark's completion of logic programs with negation as failure [14]. Integrity constraints are implications. Observations are conjunctions of literals. Integrity constraints are also used to represent, uniformly, condition action rules, commitment rules, deontic modalities, such as obligations and prohibitions, as well as goals. Goals can be properties to hold in the future. For example, an iff-definition can be used to represent the belief of an agent that,

> for the agent to have an object, it can either buy it or steal it, and there is no other way.

Moreover, integrity constraints can be given to represent

- the prohibition that penniless agents can not buy any object,
- the obligation that every time an agent buys a television (in the UK) then it should get a TV licence for it, and
- the condition-action rule that, whenever an agent who is stealing an object hears the police knocking at the door, then the agent should run away.

Then, given a goal to have a television, a possible plan for achieving this goal is to buy a television and get a TV licence for it.

Note that the above informal description is a simplification of the given domain, in that we omit a representation of time. The knowledge representation by iff-definitions and integrity constraints is sufficiently general to allow the incorporation of different temporal reasoning theories, for example the event calculus, Davila's extension of it [22] incorporating macro-events in the spirit of GOLOG, the situation calculus and plan libraries.

The reasoning engine underlying an agent combines backward reasoning with the iff-definitions, unfolding goals to subgoals, and forward reasoning with the integrity constraints and the observations, in a manner based on the abductive proof procedures of [34, 50, 64]. Planning and reactivity are achieved by reasoning backwards and forwards, respectively. Each observation is explained abductively and the explanations are used in conjunction with the beliefs.

Kowalski and Sadri's agents are reactive in two senses:

1. they recognise the need for *replanning* in the light of new observations, and
2. they can dynamically generate new goals through the activation of integrity constraints that characterise the norms of behaviour. Thus, agents have *dynamically generated goals* as well as initial goals.

In the remainder of this section we will refer to reactivity in the two senses above as REPLAN and DYNAMIC GOALS, respectively. In the above example, if an agent observes that it has no money (for example after receiving a bank statement), the earlier plan for having a TV by buying one is invalidated, and the agent has to replan. This is reactivity in the first sense described above (REPLAN). The only remaining plan is to steal a TV. Suppose that the agent executes this plan and observes the presence of the police. Then, the condition-action rule generates a new goal of running away. This illustrates reactivity in the second sense above (DYNAMIC GOALS).

Shanahan [67] represents the beliefs of agents via a formulation of the event calculus whose semantics is given by circumscription [54]. This formulation of the event calculus includes compound as well as primitive actions. For example, consider the domain of several rooms connected by doors. The connection can be represented by a set of ground facts of the form

> a given door connects two rooms.

A primitive action can be

> go through a door.

Compound actions can be defined in terms of primitive ones, for example

> the compound action "go from a room to another" can be defined in terms of the primitive action, e.g. "go through a door, connecting a room with an adjacent one".

The given formulation of the event calculus allows *hierarchical planning*, which, in turn, facilitates planning in progression order [68]. In hierarchical planning, even though a complete plan is generated, not every action in the plan needs to be fully decomposed in terms of primitive actions. A *progression planner* generates the earlier actions of a plan first. The advantages of a progression planner compared with a *regression planner* is that if the first is interrupted the first actions generated in the partially constructed plan, if primitive, can be executed immediately. This is not the case with a regression planner.

Similarly to [48], Shanahan's agents augment their beliefs by means of observations from the environment, which have to be further explained. The agents are reactive in the first sense above (REPLAN).

Baral and Gelfond [2] use extended logic programming to implement a modification of the language $\mathcal{A}$. The beliefs of the agents are expressed by theories in the language $\mathcal{A}$ together with a history of past observations. The agents' goals are properties to hold in the future.

The logic program is used proactively to reduce goals to plans. These agents are reactive in the first sense above (REPLAN). Observations from the environment are directly added to the knowledge base without further explanation and are not used as triggers for generating actions.

Language $\mathcal{A}$ is also used as the basis of the mobile robot *Diablo* of Baral et al [3]. In addition to a background theory of actions, their preconditions and effects, Baral et al formalise a collection of production rule-type sentences, called control rules, that in effect allow reactivity of agents in the second sense above (DYNAMIC GOALS).

Pereira and Quaresma [60] represent beliefs via mental states which are given by the well-founded model [76] of an extended logic program structured into the following components

- an extended logic programming formulation of the event calculus, allowing explicit negation, to reason about time and actions;
- a set of logic programming rules and integrity constraints, giving relationships between agents' attitudes in the form of meta-predicates *Bel*, *Goal* and *Int* characterising their own and other agents' beliefs, desires and intentions, respectively;
- a set of logic programming rules describing transfer of beliefs and intentions between agents and assimilation of new information.

For example, they can specify that a naive agent always accepts new information from another agent it believes to be sincere. In contrast, a cautious agent accepts new information from another agent it believes to be sincere, provided the new information does not contradict its own beliefs.

Integrity constraints are used to enforce various characteristics, for example that a property and its explicit negation do not hold at the same time, that an agent does not believe a property and its negation and that an agent does not have contradictory goals.

Observations are past events. The agents are reactive in the sense that they revise their beliefs in response to observations.

Vivid agents [66] are characterised by beliefs, in extended logic programming form, reaction rules, generalising event-condition-action rules in active databases [79], and action rules, similar to reaction rules but lacking a triggering event in their premises. Moreover, vivid agents have goals. Reaction rules encode the reactive and communicative behaviour of the agent, whereas action rules encode the proactive behaviour, as they are used to generate plans to achieve goals.

For example, in a telecommunication scenario, a vivid agent might hold the beliefs that a base transceiver station normally sends live signals to the base station, unless there is a failure, in which case the live signal is not sent. In an automated loading dock application, where forklift agents load and unload trucks, an agent might have the reaction rules that

> if an agent A receives a message from another agent B that the agent B has got hold of a box to be loaded, then agent A sends a message back acknowledging receipt of the message.

Note that this reaction rule allows to avoid overlapping between agents.

Finally, in the same setting, an agent might hold the action rule that

> if it is at some position and wants to head north then it should make a step northbound, thus finding itself at a different position, closer to its objective.

The IMPACT system [3] [71, 31, 28, 32, 27, 26] has been developed under the sponsorship of DARPA, the US Defence Advanced Research Projects Agency, within a consortium of collaborators including the University of Maryland, Bar-Ilan University, Universitaet Koblenz-Landau, Università di Milano, Technische Universitaet Wien and the U.S. Army Research Laboratory. It proposes agents whose beliefs are represented by agent programs, in the style of logic programming, and integrity constraints as well as an action base, describing the actions that individual agents can perform. Rules in agent programs generalise condition-action rules by including deontic modalities to indicate that actions are permitted, forbidden, obliged or waived. For example, an agent program can specify that

> a tank T_1 is obliged to attack another tank T_2 if the tank T_1 is obliged to proceed on a route which is being blocked by tank T_2 which belongs to the enemy.

Integrity constraints, as with other approaches, specify situations that cannot arise and actions that cannot be performed concurrently, for example that the speed of a tank should not exceed the maximum speed allowed, and two different planes cannot fly on the same route concurrently.

Alternative sets of actions can be generated in reaction to messages according to the agent programs. Different criteria for characterising such alternative sets of actions are given, and their computational complexity is examined.

Extensions of this approach incorporate temporal and probabilistic agent reasoning [26, 27] or [71, Chapter 8], [71, Chapter 9]).

INTERRAP [57, 45, 56] uses a version of the event calculus to reason about time and actions, and incorporates hierarchical planning, like [67]. It structures beliefs in three components: world model, corresponding to the history of the agent and its actions, mental model, corresponding to the event calculus theory, and social model, providing beliefs about other agents and how to interact with them. The reasoning engine is based on an abductive proof procedure, similar to [48]. It interleaves planning with the execution of partially constructed plans.

3APL [41] is a combination of imperative and logic programming. Agents in 3APL are entities with beliefs, goals and plan rules. Beliefs are in the form of sentences in a logic language (which can be full first-order logic rather than logic programming). Goals are imperative-like programs built from primitive goals, e.g. sequences of goals, in a way similar to GOLOG. Plan rules are logic

[3] `http://www.cs.umd.edu/projects/impact/`

programming-style (guarded) rules, with (primitive and composite goals) in their body.

2.1.2 Agent architectures

A common feature of most existing agent architectures is the (explicit or implicit) presence of a cycle, which in very abstract terms can be described as *"observe-think-act"*.

Kowalski and Sadri [48] adopt a resource-bounded cycle where the resource limit is a parameter that can be determined in the implementation, e.g. in terms of inference steps or some absolute time. The thinking component consists of explaining the observations, if needed, generating actions in reaction to the observations, and planning for achieving any goals the agent might have. The architecture is independent of the planning algorithm and the underlying temporal and action theory. In Davila's implementation [22] a hierarchical progression planner based on the event calculus is used. Davila adopts an event calculus-based formulation of macro-actions that has been inspired by [53].

In this agent cycle, reasoning can be interrupted both to accept inputs (observations) and to generate outputs (actions). Outputs can be not only actions generated in response to inputs received in that iteration of the cycle, but also actions generated reactively or proactively in earlier iterations.

Shanahan [67] adopts a *sense-plan-act* cycle. The sense and plan phases in each iteration of the cycle are resource-bounded in terms of resolution steps.

Baral and Gelfond [2] also adopt an explicit agent cycle of the form *observe-select goal-plan-execute* (the first action of the plan). The cycle is not resource-bounded: at the third step of the cycle a complete plan is generated for the selected goal. In later iterations of the cycle an observation may invalidate some of the preconditions of some actions in the plan, in which case replanning is needed.

The IMPACT system [71, 31, 28, 32, 27, 26] adopts a similar cycle of the form: *receive messages-find an action to execute-execute the selected action and record its result*. The cycle is not resource-bounded: all reasoning takes place and is completed within a single iteration of the agent cycle. The agents have no explicit goals and there is no planning in the conventional sense. The step of the cycle that finds an action to be executed reasons with the agent program and the other integrity constraints. In this approach (described at length in [71]), an agent can be built on top of arbitrary software (which may be legacy code). A special mechanism (code calls) is used for bridging the actual data structures in the software and their representation as logical atoms in the framework. IMPACT provides both a new theoretical foundation and a working system, building on heterogeneous database techniques.

Vivid agents [66] run a *perception-reaction* cycle and a planning activity concurrently. The cycle receives incoming messages representing both perceptions/observations and communication events, and generates reactions as dictated by the set of reaction rules. Reactions may depend on the agent's beliefs and may update them. The planning component consists of a planner and of an action component for plan execution. The planner uses a copy of the beliefs, but it does not update the beliefs.

The remaining approaches, namely [56, 45, 66, 60], have an implicit rather than an explicit cycle.

INTERRAP [56, 45, 57] has a layered architecture consisting of three layers: the lowest is the behaviour-based layer, dealing with agent reactivity, the middle one is the local planning layer, dealing with planning, and the top one is the social planning layer, dealing with multi-agent interaction. Each layer has an associated knowledge base, as described earlier. All three layers involve planning and scheduling processes. The flow of control and thus the behaviour of the agent emerges from the interaction between the individual layers. The model provides two basic protocols for the flow of control:

1. Upward activation request: when one layer is not competent to deal with a situation, it sends a request to the next layer up. The result of the processing by the next layer is reported back to the original layer.

2. Downward commitment posting: the planning and scheduling processes at different layers coordinate their activity by communicating their commitments. This allows the local planning layer to take into account partial plans devised by the social planning layer in the course of negotiating a joint plan.

In *Pereira and Quaresma*'s approach [60] the cycle is also implicit. Agents have explicit goals for which they plan using an event calculus-based theory of action and time. The agent mental state is updated after each observation.

In all the above approaches with a planning component, the concept of intention in the BDI sense is the choice of a plan amongst alternatives and is implicit rather than explicit.

2.1.3 Multi-agent systems

In this section we consider features of the approaches previously described that are specifically appropriate for multi-agent systems applications. These features include communication languages and protocols and the capability of representing and reasoning about other agents' beliefs.

Dell'Acqua, Sadri and Toni [24] extend the language of [48] by communication actions *tell* and *ask*, as additional abducible actions, and by meta-level predicates for beliefs. The model allows asynchronous communication. They illustrate the use of the extended language in specifying updates to an agent's beliefs through communication with other agents. In [25], the authors show how integrity constraints in the language proposed in [24] can be used to model communication protocols, such as the kind proposed by FIPA. For instance, the following are examples of two protocols:

- if an agent A is asked a question by another friendly agent B, and agent A can find out the answer, then it should tell the answer to agent B.
- if an agent A is asked a question by another neutral agent B, and the agent A can find the answer within ten units of time, then A should tell the answer to B as soon as this has been found, and, if A cannot find the answer within ten units of time, then it should reply with an apology.

Dell'Acqua and Pereira [23] propose a combination of the agents of Kowalski and Sadri [48] and the dynamic logic programming of [1, 52]. In the resulting framework agents can plan and react as well as change their knowledge bases and goals and attempt to change the knowledge bases and goals of others. The dynamic logic programming gives declarative and procedural semantics to the evolution of knowledge bases from state to state.

In both the IMPACT approach and in the approach of *Pereira and Quaresma* [60], agents may hold beliefs about other agents. In the IMPACT approach, agents can also exchange messages with one another. In [60], agents use extended logic programs to specify exchange of beliefs (and intentions) between agents, but not explicitly via communication. Finally, IMPACT provides added security against unauthorised updates.

The social planning layer in *INTERRAP* [57] involves communication and reasoning about other agents' beliefs.

Hindriks et al [42] extend the single agent language 3APL with communication primitives to represent multi-agent systems. The new primitives are partitioned in two sets, one consisting of *tell* and *ask*, the other consisting of *req* and *offer*. The semantics of each set of the new primitives is based on deduction and abduction, respectively. Deduction is used to derive information from received messages, and abduction is used to obtain proposals in reply to requests.

Their communication primitives are synchronous, that is both the sender and the receiver have to participate at the same time. Unlike speech act theory, the semantics of Hindriks et al is not concerned with the mental states of speakers. The agent that *tells* something does not necessarily have to believe it, and the way that the receiving agent deals with the message is not specified in the semantics.

CommonRules is a prototype developed within EECOMS, a $29 Million, 3 year project including a consortium of collaborators in industry (IBM, Boeing, Baan, AT&T, TRW) and at US universities (University of Maryland, University of Michigan, New York University, University of North Carolina, University of Florida). The prototype emerges from experience in developing industrial applications, and it identifies rules, in general, as useful in applications such as multi-agent electronic commerce. The collaborators have identified logic programs, in particular, as a core declarative knowledge representation language, to have major advantages compared to other rule-based approaches (as well as compared to non-rule programming approaches). Communicating logic programs between multiple agents, however, raises several challenges. First, a translation is needed between heterogeneous rule systems. This involves syntactic parsing, as well as deep declarative semantics. Second, facilities for conflict handling are needed when merging and updating rule sets, for example when an agent imports/assimilates rules from other agents or when multiple business-domain experts specify and maintain rule sets.

CommonRules incorporates conflict handling and inter-agent communication facilities, that complement a variety of existing rule execution engines. In particular, CommonRules adopts a prioritised handling of conflicts, based on courteous logic programming [37]. This is a form of prioritised logic programming that can be tractably compiled (e.g., via a preprocessor) to (semantically equivalent) ordinary logic programming. Agent communication includes translations of rules/facts via a common “knowledge sharing” interlingua in XML called Business Rules Markup Language.

Carbogim and Robertson [10] develop a method for contract-based negotiation between agents, where contracts are represented as logic programs that change dynamically as a result of negotiation, until an agreement between the parties involved is reached. The revised logic programs are generated by means of argumentation techniques, adapted from those developed in [4, 29].

Poole [62] proposes a rather different, but complementary approach to multi-agent systems. He does not specify beliefs or goals for individual agents. In his approach, an agent owns a set of alternatives, each being a set of possible actions. Then, a multi-agent system consists of

- A finite set of single agents.
- A special agent, modelling the environment (called "nature" by Poole), with an associated set of alternative sets of actions.
- For each alternative owned by nature, a probability distribution over the actions of the alternative. The probability distributions give a measure of what actions are chosen by the environment.
- A logic program with negation as failure and abduction, with the (undefined) abducibles being atomic actions owned by agents. The logic program axiomatises the (causal) relationships between actions and states resulting from agents performing actions, and the assignment of (quantitative) utilities/payoffs to states. Each state corresponds to the (unique) canonical model of the logic program extended by some set of selected actions.

Poole adopts conventional (quantitative) decision-theoretic techniques to infer strategies for the choice of actions by individual agents. Given a (pure or stochastic) strategy for every agent in the multi-agent system, the overall utility for an individual agent can be computed by means of decision-theoretic techniques.

For example, a multi-agent system can consist of a footballer, a goal-keeper and the environment. A footballer can kick the ball either left or right (but not both), and a goal-keeper can jump either left or right (but not both) in order to stop the ball. Every time a footballer is in front of the goal-keeper, the footballer must decide in which direction to kick the ball, and the goal-keeper must decide in which direction to jump. The logic program can specify which agents' actions lead to a world in which a goal is scored, and the utility of such world for footballer (high) and goal-keeper (low). Probabilities of (fictitious) actions by the environment can be given to express that the chances of a goal to be scored is quite high if the goal-keeper jumps in some direction and the footballer kicks in exactly the same direction.

Bozzano et al [5] provide a logic programming-based framework for developing multi-agent systems, where agents are equipped with sets of facts defining their state and sets of rules specifying their behaviour. Agents are implemented in a Prolog-like language, enriched with primitives for updates and communication.

2.2 Agent-oriented logic programming languages

In this section we illustrate some proposed logic programming-based languages for multi-agent systems, as well as some of their applications. Such languages extend Prolog by exhibiting some of the following features:

- multi-threading (a thread can be thought of as a process, but multiple threads can share the same execution space);
- inter-thread communication;
- condition-action/reaction rules;
- code mobility over a computer network.

Qu-Prolog

Qu-Prolog [15, 16] is a logic programming language, which is an extension of Prolog, and is primarily intended for the implementation of interactive theorem provers. As such, it is suitable for the development of distributed artificial intelligence applications, such as interacting agents. It contains all the features expected of a standard Prolog, such as data base manipulation and meta-level predicates. In addition, its latest version, version 5.0, offers multi-threading and inter-thread communication facilities. Every Qu-Prolog process has the ability to launch threads thus providing concurrency in the execution.

An agent's beliefs can be characterised by a dynamic database which can be accessed and updated by all the agent's threads. The dynamic database thus provides a blackboard mechanism for inter-thread communication.

In the agent setting, the multi-threading facility can be exploited, for example, to perform belief revision by incorporating a component that constantly monitors the updates added to the agent's beliefs by other agents, and a background component that derives and asserts further interesting new consequences of the updates. The multi-threading facility also allows concurrent execution of queries against an agent's database.

Threads can explicitly communicate with one another by sending messages, which are Prolog terms. When messages are sent the recipient will also receive information about where the message originated from and where a reply should be sent.

Threads are equipped with sets of reaction rules specifying how threads should react to the messages they receive. The threads execute the reaction rules repeatedly. These reaction rules can take the following form:

$\texttt{message\_choice(}$
$\texttt{message\_pattern}_1 \rightarrow \texttt{action\_conjunction}_1\texttt{;}$
$\texttt{message\_pattern}_2 \rightarrow \texttt{action\_conjunction}_2\texttt{;}$

```
...
message_pattern_k →action_conjunction_k).
```
Communication can take place amongst threads executing on several processes on two machines. It is this aspect, communications amongst several machines, that exploits the power of a computer network.

Jinni

Jinni (Java INference engine and Networked Interactor) [?, 74] is a commercial multi-threaded logic programming language for integrating Prolog-based knowledge processing components and Java objects in distributed applications, such as networked client/server applications and applets over the Internet. Coordination between threads is achieved via blackboards, local to threads, and synchronisation via term unification. Threads and the computation they perform can be mobile. Jinni is not equipped with any high-level communication primitives.

Jinni aims at being a convenient development platform for distributed artificial intelligence applications in general, and for building intelligent autonomous agent applications in particular. It has been used, for example, to develop a stock market simulator, with user programmable intelligent agents. Connection of the simulator to the real world Internet-based stock trade services is under development.

One of the possible future applications envisaged for Jinni is assistance of customers of "intelligent" appliances, e.g. web-based TV, mobile cell phones, car computers.

Jinni's multi-threading and networking model have recently been ported to BinProlog [73].

Ciao Prolog

Ciao Prolog [40, 17] is a public domain constraint logic programming system subsuming ISO-Prolog. It has a number of useful features for the implementation of agent-based software. Ciao Prolog supports threads, a shared database with atomic updates, objects, and high-level remote inter-process communication (including interfaces for local and remote databases [18] and WWW [8] access). The Ciao module/package system [9] makes it possible to add syntactic and semantic extensions to the language, in a more structured way than traditional Prolog systems, and this can be done without affecting other parts of the language. Ciao also includes a large set of libraries. Additionally, Ciao Prolog generates multiplatform executables based on compilation to bytecode and allows movement of code dynamically across platforms.

Ciao Prolog allows goals to be launched, by means of high-level primitives, in separate threads. These threads can communicate and synchronise by using the Prolog database, which is shared among the threads [12]. Threads accessing a shared fact can (optionally) wait for it instead of failing if it does not exist yet. As an example, the following code:

```
:- concurrent current_temp/1.

monitor(Temp) :-                  read_temp:-
    read_temp &,                      read_temp_device(Temp),
    current_temp(Temp).               assert(current_temp(Temp)).
```

will launch a concurrent thread for the predicate `read_temp`, using the `&/1` primitive, and will wait for the `concurrent` predicate `current_temp/1` to exist, while `read_temp/0` accesses a (possibly external) source of temperature measures and asserts a `concurrent` fact stating the read temperature. Other primitives are available: an extension of the `&/1`, in conjunction with the `@/2` (*placement*) operator, generates a *handle* to a concurrent goal [7, 39, 12] which can be executed in a remote machine. These basic primitives can be used as building blocks to model a variety of knowledge representation paradigms and architectural models.

Condition-Action Rules [47] can be implemented in Ciao as a language extension using the capabilities of the language syntax packages. They can be used to specify the reaction of programs to (monotonic) updates to the database.

Ciao Prolog supports objects [61], each having its own state represented by a set of *private* facts which are not accessible outside the object. Objects are fully integrated with the concurrent/distributed primitives, and objects can execute in their own thread and in a distributed fashion. Objects share their code with other objects of the same class, but they do not share the set of internal facts which model their private state (e.g., beliefs). These facts can be concurrently and atomically updated and accessed.

Ciao Prolog also allows a special kind of object, called active, which can be viewed as network-wide servers of methods. An active object is launched (possibly remotely) as a daemon, and answers the requests (method invocations) it receives. Active objects can be used to implement distributed knowledge bases (or blackboards) [7] in which updates to global facts are automatically available to remote agents.

Eel

Eel [20] uses an event based approach to process communication, and is based on a limited model of communication which only allows synchronous point to point communication. It does not have the concepts of sending and receiving messages, but, instead relies on message unification as a method of exchanging information between concurrent processes. Its syntax is similar to Prolog, and it shares Prolog's operational semantics except for the special reserved predicates that deal with the synchronisation of communication.

GAEA

GAEA [4] [59] is a programming language extending Prolog by means of the following features:

- modules (called *cells*) and inheritance between them;
- multi-threading;
- thread communication via shared memory (called *cell variables*), allowing synchronisation between threads;
- partially specified terms, that can allow to determine the structure of modules at run-time.

Moreover, GAEA is equipped with a built-in (X-window) interface and a network interface for a distributed, multi-processing environment.

In GAEA, threads execute a logic program within an environment, given by a set of modules. Modules themselves can be seen as fragments of logic programs. Inheritance and overriding between modules apply.

GAEA has been used to give a dynamic version of Brooks' subsumption architecture [6] where symbol manipulation is allowed to permit communication between agents, and dynamic reconfiguration is achieved (see [58]). The GAEA-based dynamic subsumption architecture has been used to implement football playing agents in a RoboCup-like environment [58].

2.3 Implementation and applications

In this section we briefly describe the implementation of the architectures and systems we have surveyed earlier and describe some of the applications they have been used for.

Kowalski and Sadri's architecture [48] has been implemented in Prolog [22] and Java/Voyager [11], and tested on a number of applications. Three of these applications are particularly noteworthy:

- Distributed information integration [78], whereby agents are equipped with knowledge that allows them to act as mediators that integrate the information sources so as to answer user queries more effectively, to handle heterogeneity between different sources and to facilitate the resolution of conflicts that might arise between them.

[4] `http://ci.etl.go.jp/gaea`

- Collective management of information networks [75], e.g. the WWW, whereby agents can be employed to check and enforce structural properties of the networks (e.g. that there are no dangling links) as well as properties on the content of nodes in the network.

- Learning agents [77], for example those that learn preconditions of actions.

The approach of *Shanahan* [67] has been implemented in Prolog and tested on navigation problems using a Khepera robot. The robot has (proximity) sensors and wheels, so that it can move around an environment with obstructions, for example an office-like environment. The architecture allows the robot to sense its environment, to plan its moves for getting around, to execute the moves and to replan in the presence of newly observed obstructions.

The architecture of *Baral and Gelfond* [2] is implemented in Prolog and XSB-Prolog [13] and it is envisaged for an application to shuttle control.

Diablo [3] has been used as the supporting architecture for mobile robots competing in two AAAI robot competitions. In the first competition, the AAAI'96 robot contest, robots were required to navigate their way round an office-like environment, given a topological map of it, and without hitting any obstacles. Diablo was placed third in the finals. The second competition, the AAAI'97 contest, was a vacuum cleaning contest, where the robots were required to navigate their way around a corridor and several rooms in order to clean the rooms. The robot had to work under various constraints, for example it was forbidden to enter a room when a human was in it, but they could clean the corridor in the presence of humans. Diablo won the first place in this contest.

The system of *Pereira and Quaresma* [60] has been implemented using the REVISE system [21] and XSB-Prolog [13], and tested on examples.

Vivid agents [66] have been implemented in PVM-Prolog [19] and tested on distributed diagnosis problems, such as distributed diagnosis of a computer network and diagnosis of unreliable communication protocols. In such applications, a centralised technique is costly and inefficient, and it can cause large communication overheads. The Vivid agents approach decomposes the system into a set of subsystems, each of which is diagnosed by an agent which has detailed knowledge over its subsystem and a more abstract knowledge of the neighbouring systems.

The core of IMPACT [71] has been implemented in Java. The system is built on a collection of heterogeneous databases. It has been tested on a number of applications, two of which are nontrivial U.S. Army applications:

- War reserves planning by agencies responsible for the integration of army logistics. This involves massive amounts of data distributed over many databases.

- Distributed simulations of route and manoeuvre planning over free terrain. This application involves tracking enemy vehicles on the battle field and attempting to predict what enemy agents are likely to do in the future, based on meta knowledge about them. The system is intended for training and simulation purposes rather than deployment on the battle field.

INTERRAP [57, 45, 56] has been implemented in Prolog and Oz [70] and tested on a number of real-life software and hardware applications:

- Robot control for automated loading dock, whereby simple forklift agents are responsible for loading tracks in a coordinated manner, avoiding conflicts between one another.

- Transportation task scheduling on a railroad network.

- Computer integrated manufacturing.

The *CommonRules* prototype [5] [38] has been implemented in Java. It has been applied on inter-enterprise integration in manufacturing supply chains and on electronic commerce. In particular, it has been applied to:

- Multi-agent negotiation that includes communicating, and sometimes revising, contractual agreements that describe bids for goods (and services). A bid contains a contractual agreement that specifies not just pricing and product attributes of a good, but also the more complex terms and conditions of the proposed deal.

- Security authorisation policies that include delegations between multiple agents (e.g. with cryptographically-authenticated certificates).

These applications make use of CommonRules' features for agent communication and prioritised handling of conflicts.

Poole's approach [6] [62] has been implemented in Prolog and tested on robotics examples.

The framework of *Bozzano et al* [5] has been applied for

[5] `http://www.research.ibm.com/rules/`
[6] `http://www.cs.ubc.ca/spider/poole/`

- train scheduling, in collaboration with the Italian Railway Company;
- planning of goods transportation, in collaboration with Elsag Baley, an international company which provides service automation;
- retrieval of medical information contained in medical databases.

3 Research challenges

The survey in section 2 illustrates the diverse issues that have so far been addressed in computational logic-based agent technology. However, there remain many more research and development challenges. These can be roughly divided in three areas:

- single agent architectures,
- multi-agent systems,
- implementation and application development.

3.1 Single agent architectures

The functionalities of single agent architectures can be enhanced significantly by further research on a number of areas including the following:

- Sophisticated and efficient planners. Traditional artificial intelligence planners plan from first-principle, decomposing a goal to sub-goals progressively, until a complete plan in the form of a sequence of actions is found. Agents have to act in a timely fashion in a dynamic environment about which they may have incomplete information. They will be handicapped if they cannot act until they have formed a complete plan. Therefore, sophisticated planning techniques will have to be found, that can construct plans hierarchically and incrementally, and that can deal efficiently with changes in the environment requiring replanning. Furthermore, there is a need for developing plan libraries and plan repair techniques to avoid having to recompute plans for similar goals.
- Integration of proactivity and reactivity. The need for this integration has been already recognised by many researchers and practitioners. However, more sophisticated techniques are needed for the selection of goals and actions to ensure timely responses. Existing techniques for prioritisation as well as computing and maximising utility of actions would have to be further developed to schedule and interleave execution of plans and reactions to the environment.

- Belief revision. Agents have to be able to revise their beliefs and goals in the light of new information that they observe or that they receive by communication by other agents (in a multi-agent setting). There is a great deal of research regarding belief revision, for example in knowledge bases. These techniques would have to be tested and possibly further developed and adapted for agent systems.

- Perception. In almost any application agents need to perceive the aspects of the environment that are important to their functioning. They have to be selective in their perception to avoid being overwhelmed by too many stimuli. Their functioning would be jeopardised if there are delays in their perception of the environment. This issue arises both for hardware and software agents.

- Learning. Given that agents have to function in complex evolving environments, a capacity to learn will improve their performance. They can learn plans that have been useful in the past, preconditions of actions, profile of users, communication protocols and so on. Moreover, computational logic has made significant contributions in learning through inductive logic programming [51]. Incorporation of such a learning tool within existing agent architectures is a worthy challenge.

3.2 Multi-agent systems

Further work is needed on a number of areas, including the following:

- Cooperative and joint planning. Agents can have joint as well as individual goals. Cooperative behaviour becomes crucial for achievement of joint goals and can help in the achievement of individual goals.

- Formalisms and reasoning techniques for negotiation. Negotiation plays an important part when agent have joint goals, to construct joint plans, and when they have conflicting individual goals, to resolve the conflicts. Contract-based negotiation and argumentation seem to be promising in realising negotiation within a computational logic setting.

- Further development of languages and tools for representing and reasoning about other agents' beliefs, goals and intentions. To cooperate and to negotiate, an agent in a multi-agent setting can be at an advantage if it has beliefs about the other agents' beliefs, goals and intentions. The performance of a community of such agents can be greatly enhanced: agents can avoid proposing actions and plans that they know would be unacceptable to others and they can preempt other agents. Furthermore, agents can behave more effectively if they have information about the intentions of other agents thus increasing the chances of success of their actions.

- Development of communication languages, protocols and their semantics. Language allows cooperation, negotiation and effective dissemination of information. There are already agent systems that have attempted to incorporate communication. These would have to be further developed.

3.3 Applications

Many of the approaches described earlier in this report have reached sufficient maturity to be tested on more complex, realistic applications. A number are currently being used in large-scale industrial applications, such as INTERRAP, `IMPACT` and CommonRules. Many industrial and commercial concerns have identified the possible benefits of agent technologies: we discuss this in more detail in the next section.

4 Future opportunities

Industry and commerce are rapidly beginning to appreciate the potential of agent technologies for a number of applications. Jennings, Sycara and Wooldridge [43] identify four key domains:

- industrial: manufacturing, process control, telecommunication, air-traffic control, transportation systems;
- commercial: information management, electronic commerce, business process management;
- entertainment: games, interactive theatre and cinemas;
- medical: patient monitoring, health care.

A number of companies have established agent technology divisions which are often in direct collaboration with academia. The purpose of such divisions is to develop and exploit as fast as possible the emergent agent technology.

The emerging consensus from such research and development endeavour is the recognition of the necessary complexity of the systems, the need to handle rich and diverse sources of information and complex processes, and the need to adjust to increasingly sophisticated user demands. This leads to the realisation of the importance of incorporating intelligence in the systems. This is one of the major aspects where computational logic-based agents can play a significant role. As the survey of these agent-based systems in the earlier section 2 indicates, these systems are designed to deal with rich representations of knowledge and to incorporate proactive and reactive behaviour in dynamic environments. This poses an exciting opportunity and a significant challenge to this research community to make industry aware of the research developments and to demonstrate the promise of these techniques in dealing with the emerging problems. Various projects have already started and indeed made progress in doing this, for example IMPACT, CommonRules and Jinni. It is noteworthy that these systems are primarily funded and developed in the US, despite the fact that computational logic originated and enjoyed its earlier developments in Europe, and that even now a great deal of the research in computational logic-based agents is being carried out at European academic establishments.

In addition to opportunities that are immediately open to the field in industrial and commercial exploitation, there is a great deal of scope for furthering the research. Promising developments in computational logic for argumentation and defeasible reasoning have reached sufficient maturity so that they can be embedded within agent architectures. This would significantly enhance the functionalities of these architectures and pave the way to a number of exciting applications such as business-to-client and business-to-business electronic commerce.

Agents have to work in dynamic environments and need to adjust to increasingly sophisticated user demands. Incorporating a learning facility within agent systems would enhance its functions.

The symbolic manipulation of data and the combination of reactivity and proactivity in computational-logic based agents show promise for information integration in dynamic domains such as the WWW.

The incorporation of user profiling techniques within agent architectures could provide fine-tuned and personalised access to heterogeneous information sources and agent functionalities.

As the agent-oriented programming languages develop we will have further opportunities for faster and easier application development.

5 Conclusions

We have surveyed current work in the area of computational logic-based agents, and emphasised that such work includes not only specification of agents but computational models, implementations and more or less realistic applications all within a unified framework, thus filling the gap between theory and practice that often arises in work in agents.

We have argued that computational logic, together with developments in distributed artificial intelligence, shows great promise in enriching the research in agents. In particular, the emerging consensus from such research and development endeavour is the recognition of the necessary complexity of the systems, the need to handle rich and diverse sources of information and complex processes, and the need to adjust to increasingly sophisticated user demands, thus leading to the realisation of the importance of incorporating intelligence in the systems. This is a significant aspect to which computational logic-based agents can contribute.

The development of agents via computational logic techniques is a young but fast growing area. The research community is enthusiastic and the interest of industry is also growing, especially in the United States. Indeed, only during this last year, there have been two workshops organised under the auspices of CompulogNet, and both of them exceeded the expectations of the organisers in the number and the enthusiasm of the participants.

Acknowledgements

The authors would like to thank Alastair Bert, Jürgen Dix, Manuel Hermenegildo, Luis Moniz Pereira and Silvana Zappacosta Amboldi for useful comments and contributions to this report, and to Jim Cunningham, Abe Mamdani and Jeremy Pitt for helpful discussions.

References

[1] J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinski, Dynamic logic programming. In A. Cohn, L. Schubert (eds.), *Proc. KR'98*, International Conference on Principles of Knowledge Representation and Reasoning, pages 98–109, Morgan Kaufmann, 1998.

[2] C. Baral, M. Gelfond, Reasoning Agents in Dynamic Domains. In J. Minker (ed.) *Pre-Proc. of Workshop on Logic-based Artificial Intelligence*, Washington D.C., June 1999.

[3] C. Baral, L. Floriano, A. Hardesty, D. Morales, N. Nogueira, T.C. Son, From Theory to Practice: the UTEP Robot in the AAAI96 and AAAI97 Robot Contests. *Proc. Agents'98*, Second International Conference on Autonomous Agents, Minneapolis/St. Paul, May 9–13, 1998.

[4] A. Bondarenko, P.M. Dung, R.A. Kowalski, F. Toni, An Abstract, Argumentation-theoretic Framework for Default Reasoning. *Artificial Intelligence* 93 (1,2): 63–101, 1997.

[5] M. Bozzano, G. Delzanno, M. Martelli, V. Mascardi, F. Zini, Multi-agent Systems Development as a Software Engineering Enterprise. *Proc. of First International Workshop on Practical Aspects of Declarative Languages* (PADL'99), (G. Gupta ed.) Springer Verlag, LNCS 1551, pages 46-60, 1999.

[6] R. Brooks, Intelligence without Representation. *Artificial Intelligence* 47:139–160, 1991.

[7] D. Cabeza and M. Hermenegildo. Implementing Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the AGP'96 Joint Conference on Declarative Programming*, pages 67–78, 1996.

[8] D. Cabeza and M. Hermenegildo. WWW Programming using Computational Logic Systems (and the PILLOW/CIAO Library). In *Proc. of the Workshop on Logic Programming and the WWW*, in conjunction with WWW6, San Francisco, CA, April 1997.

[9] D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *Proc. of the Workshop on Parallelism and Implementation of (C)LP Systems.* In conjunction with ICLP'99, New Mexico State University, US, December 1999.

[10] D. Carbogim, D. Robertson, Contract-based Negotiation via Argumentation (Preliminary Report). In *Proc. of the Workshop on Multi Agents Systems in Logic Programming.* In conjunction with ICLP'99, New Mexico State University, US, December 1999.

[11] Y. Carer, *Distributed Communicating Agents.* MEng Thesis, Department of Computing, Imperial College of Science, Technology and Medicine, London, UK, 1999.

[12] M. Carro and M. Hermenegildo. Concurrency in Prolog Using Threads and a Shared Database. In *Proc. ICLP'99*, International Conference on Logic Programming, MIT Press, 1999.

[13] W. Chen, D.S. Warren, Tabled Evaluation with Delaying for General Logic Programs. *JACM* 43(1):20–74, 1996.

[14] K.L. Clark, Negation as Failure. In H. Gallaire, J. Minker (eds.), *Logic and Data Bases*, pages 293–322, Plenum, NY, New York, 1978.

[15] K.L. Clark, P. Robinson, R. Hagen, Multi-threading and Message Communication in Qu-Prolog. In *Proc. of the Australian Workshop on Distributed Artificial Intelligence*, Brisbane, 1998.

[16] K.L. Clark, P. Robinson, R. Hagen, Programming Internet Distributed DAI Applications in Qu-Prolog. In *Proc. of the Australian Workshop on Distributed Artificial Intelligence*, Brisbane, 1998.

[17] The CLIP Group. CIAO System Reference Manual. The Ciao System Documentation Series–TR CLIP3/97.1, Facultad de Informática, UPM, August 1997.

[18] J.M. Gomez, D. Cabeza, and M. Hermenegildo. persdb: Persistent Database Interface. Technical Report D3.1.M2-A1 CLIP9/98.0, RADIOWEB Project, December 1998.

[19] J.C. Cunha, R.F.P. Marques, Distributed Algorithm Development with PVM-Prolog. *Proc. 5th Euromicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society Press, 1997.

[20] T.S. Dahl, The Eel Programming Language and Internal Concurrency in Logic Agents. In *Proc. of the Workshop on Multi Agents Systems in Logic Programming*. In conjunction with ICLP'99, New Mexico State University, US, December 1999.

[21] C. Damasio, W. Negdl, L.M. Pereira, Revise: an Extended Logic Programming System for Revising Knowledge Bases. In *KR'94*, International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, 1994.

[22] J. Davila. *Agents in Logic Programming*. PhD thesis, Imperial College of Science, Technology and Medicine, London, UK, 1997.

[23] P. Dell'Acqua, L. M. Pereira, Updating Agents. In *Proc. of the Workshop on Multi Agents Systems in Logic Programming*. In conjunction with ICLP'99, New Mexico State University, US, December 1999.

[24] P. Dell'Acqua, F. Sadri, F. Toni, Combining Introspection and Communication with Rationality and Reactivity in Agents. In U. Furbach, L. Farinas del Cerro (eds.), *Proc. of JELIA'98* 6th European Workshop on Logics in Artificial Intelligence, pages 17–32, LNAI 1489, Springer Verlag, 1998.

[25] P. Dell'Acqua, F. Sadri, F. Toni, Communicating Agents. In *Proc. of the Workshop on Multi Agents Systems in Logic Programming*. In conjunction with ICLP'99, New Mexico State University, US, December 1999.

[26] J. Dix, S. Kraus, V.S. Subrahmanian, Temporal Agent Reasoning. Technical Report, Department of Computer Science, University of Maryland, September 1999.

[27] J. Dix, M. Nanni, V.S. Subrahmanian, Probabilistic Agent Reasoning. Technical Report, Department of Computer Science, University of Maryland, September 1999.

[28] J. Dix, V.S. Subrahmanian, G. Pick, Meta Agent Programs. To appear in *Journal of Logic Programming*, 2000.

[29] P.M. Dung, The Acceptability of Arguments and its Fundamental Role in Non-monotonic Reasoning and Logic Programming and N-Person Game. *Artificial Intelligence* 77: 321–357, 1995.

[30] T. Eiter, V.S. Subrahmanian, G. Pick, Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence* 108(1–2):179–255, 1999.

[31] T. Eiter, V. S. Subrahmanian, Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence* 108(1–2):257–307, 1999.

[32] T. Eiter, V.S. Subrahmanian, T.J. Rogers, Heterogeneous Active Agents, III: Polynomially Implementable Agents. Technical Report, Institut für Informationssysteme, Technische Universität Wien, INFSYS RR-1843-99-07, May 1999.

[33] K. Fisher, Agent-Based Design of Holonic Manufacturing Systems. To appear in *Journal of Robotics and Autonomous Systems*, Elsevier Science B.V., 1999.

[34] T.H. Fung, R.A. Kowalski, The Iff Proof Procedure for Abductive Logic Programs. *Journal of Logic Programming* 33(2):151–165, Elsevier, 1997.

[35] M. Gelfond, V. Lifschitz, Logic Programs with Classical Negation. In *Proc. ICLP'90*, 7th International Conference on Logic Programming, pages 579–597, MIT Press, 1990.

[36] M. Gelfond, V. Lifschitz, Representing Action and Change by Logic Programs. *Journal of Logic Programming* 17 (2,3,4):301–322, 1993.

[37] B.N. Grosof, Prioritised Conflict Handling for Logic Programs. In J. Maluszynski (ed.) *Proc. ILPS'97*, International Logic Programming Symposium, pages 197–211, MIT Press, 1997.

[38] B.N. Grosof, Y. Labrou, H. Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In M.P. Wellman (ed.), *Proc. of EC'99*, 1st ACM Conference on Electronic Commerce, ACM Press, 1999.

[39] M. Hermenegildo, D. Cabeza, and M. Carro. Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems. In *Proc. ICLP'95*, 12th International Conference on Logic Programming, pages 631–645, MIT Press, 1995.

[40] M. Hermenegildo, F. Bueno, D. Cabeza, M. García de la Banda, P. López, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, 1999.

[41] K.V. Hindriks, F.S. de Boer, W. van der Hoek and J.-J.Ch. Meyer, Control Structures of Rule-Based Agent Languages. *Proc. ATAL'98*, Fifth International Workshop on Agent Theories, Architectures, and Languages, LNAI 1555, Springer Verlag, 1998.

[42] K. V. Hindriks, F. S. de Boer, W van der Hoek, J-J Ch. Meyer, Semantics of Communicating Agents Based on Deduction and Abduction. In F. Dignum, B. Chaib-draa (eds.), *Proc. of the Workshop on Agent Communication Languages.* In conjunction with IJCAI'99, pages 105–118, 1999.

[43] N.R. Jennings, K. Sycara, M. Wooldrige, A Roadmap of Agent research and Development. *Autonomous Agents and Multi-Agent Systems* 1:7–38, Kluwer Academic Publishers, 1998.

[44] C.G. Jung, Situated Abstraction Planning by Abductive Temporal Reasoning. In H. Prade (ed.), *Proc. ECAI'98*, 13th European Conference on Artificial Intelligence, pages 383–387, Wiley, 1998.

[45] C.G. Jung, K. Fischer, A Layered Agent Calculus with Concurrent, Continuous Processes. In M. P. Singh, A.S. Rao, M. J. Wooldridge (eds.), *Intelligent Agents IV*, LNAI 1365, Springer Verlag, 1998.

[46] A.C. Kakas, R.A. Kowalski, F. Toni, The Role of Abduction in Logic Programming. In D.M. Gabbay, C.J. Hogger and J.A. Robinson (eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming* 5:235–324, Oxford University Press, 1998.

[47] R. A. Kowalski. Logic Programming with Integrity Constraints. In *Proc. JELIA*, European Workshop on Logics in Artificial Intelligence, pages 301–302, 1996.

[48] R. A. Kowalski, F. Sadri. From Logic Programming towards Multi-agent Systems. To appear in *Annals of Mathematics and Artificial Intelligence*, 1999.

[49] R.A. Kowalski, M.J. Sergot, A Logic-based Calculus of Events. *New Generation Computing* 4:67–95, 1986.

[50] R.A. Kowalski, F. Toni, G. Wetzel, Executing Suspended Logic Programs. *Fundamenta Informatica* 34:1–22, 1998.

[51] N. Lavrac, Computational Logic and Machine Learning: a Roadmap for Inductive Logic Programming. Technical Report, J. Stefan Institute, Ljubljana, Slovenia, 1998.

[52] J.A. Leite, L.M. Pereira, Iterated Logic Program Updates. In J. Jaffa (ed.), *Proc. JICSLP'98*, Joint International Conference and Symposium on Logic Programming, pages 265–278, MIT Press.

[53] H.J. Levesque, R. Reiter, Y. Lesperance, F. Lin, R.B. Scherl, GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, Special Issue in Reasoning about Action and Change, 31(1–3):59–83, 1997.

[54] J. McCarthy, Circumscription – a Form of Non-monotonic Reasoning. *Artificial Intelligence* 13:27–39, 1980.

[55] J. McCarthy, P.J. Hayes, Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer, D. Michie (eds.), *Machine Intelligence* 4:463–502, Edinburgh University Press, 1969.

[56] J.P. Mueller, *The Design of Intelligent Agents: a Layered Approach.* LNAI 1177, Springer Verlag, 1996.

[57] J.P. Mueller, K. Fischer, M. Pischel, A Pragmatic BDI Architecture. In M.N. Huhns, M. P. Singh (eds.), *Readings in Agents*, pages 217–225, Morgan Kaufman, 1998.

[58] H. Nakashima, I. Noda, Dynamic Subsumption Architecture for Programming Intelligent Agents. *Proc. of the International Conference on Multi-agent Systems*, pages 190–197, AAAI Press, 1998.

[59] I. Noda, H. Nakashima, K. Handa, Programming Language GAEA and its Application for Multi-agent Systems. In *Proc. of the Workshop on Multi Agents Systems in Logic Programming.* In conjunction with ICLP'99, New Mexico State University, US, December 1999.

[60] L. M. Pereira, P. Quaresma, Modelling Agent Interaction in Logic Programming. In S. Fukuda (ed.), *Proc. INAP'98*, 11th International Conference on Applications of Prolog, Tokyo, Japan, 1998.

[61] A. Pineda and M. Hermenegildo. O'ciao: An Object Oriented Programming Model for (Ciao) Prolog. Technical Report CLIP 5/99.0, Facultad de Informática, UPM, July 1999.

[62] D. Poole, The Independent Choice Logic for Modelling Multiple Agents under Uncertainty. *Artificial Intelligence* 94(1-2):7–57, 1997.

[63] A.S. Rao M.P. Georgeff, Modelling Rational Agents within a BDI-architecture. In R. Fikes, E. Sandewall (eds.), *Proc. KR'92*, International Conference on Knowledge Representation and Reasoning, pages 473–484, Morgan Kaufmann, 1991.

[64] F. Sadri, F. Toni, Abduction with Negation as Failure for Active Databases and Agents. *Proc. AI*IA 99*, 6th Congress of the Italian Association for Artificial Intelligence, pages 353–362, Pitagora Editrice Bologna, 1999.

[65] M. Schroeder, An Efficient Argumentation Framework for Negotiating Autonomous Agents. In *Proc. MAAMAW'99*, Workshop on Modelling Autonomous Agents in a Multi-Agent World, Springer Verlag, 1999.

[66] M. Schroeder, G. Wagner, Vivid Agents: Theory, Architecture, and Applications. Technical Report, Department of Computer Science, City University, London, UK, 1999.

[67] M.P. Shanahan, Reinventing Shakey. *Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics*, pages 125–135, 1998.

[68] M.P. Shanahan, An Abductive Event Calculus Planner. To appear in *Journal of Logic Programming*, 2000.

[69] Y. Shoham, Agent-oriented Programming. *Artificial Intelligence* 60:51–92, 1993.

[70] G. Smolka, The Oz Programming Model. In J. VanLeeuwen (ed.) *Computer Science Today*, LNCS 1000, Springer Verlag, 1995.

[71] V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, R. Ross, *Heterogeneous Agent Systems*. MIT Press, 2000. (See `http://www-mitpress.mit.edu/book-home.tcl?isbn=3D0262194244` or `http://www.cs.umd.edu/users/dix/web_version.ps.gz`)

[72] P. Tarau, Logic Programming Tools for Advanced Internet Programming. In J. Maluszynski (ed.), *Proc. ILPS'97*, International Symposium on Logic Programming, pages 33–34, MIT Press, 1997.

[73] P. Tarau, The BinProlog Experience: Implementing a High-Performance Continuation Passing Prolog Engine. BinNet Corp. Technical Report, 1998. `http://www.binnetcorp.com/BinProlog/index.html`

[74] P. Tarau, Jinni: Intelligent Mobile Agent Programming at the Intersection of Java and Prolog. In *Proc. PAAM'99*, Fourth International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, pages 109–121, 1999.

[75] F. Toni, Automated Reasoning for Collective Information Management. In *Proc. LocalNets*, International Workshop on Community-based Interactive Systems, in conjunction with AC'99, the Annual Conference of the EC I^3 Programme, 20–22 October 1999, Siena, Italy.

[76] A. Van Gelder, K.A. Ross, J.S. Schlipf, Unfounded Sets and the Well-founded Semantics for General Logic Programs. In *Proc. ACM SIGMOD-SIGACT, Symposium on Principles of Database Systems*, 1988.

[77] B. Wellner, *Concept Learning in Multi-agent Systems.* MSc Thesis, Department of Computing, Imperial College of Science, Technology and Medicine, London, UK, 1999.

[78] I. Xanthakos, *Logic-agent Based System for Semantic Integration of Information.* MSc Thesis, Department of Computing, Imperial College of Science, Technology and Medicine, London, UK, 1999.

[79] C. Zaniolo, S. Ceri, C. Faloutsos, R. T. Snodgrass, V.S. Subrahmanian, R. Zicari, *Advanced Database Systems.* Morgan Kaufman, 1997.