Computational Logic and Agents
Miniscuola WOA 2009

Viviana Mascardi

University of Genoa – Department of Computer and Information Science

July, 8th, 2009

Outline

Outline

Computing with logic?

Computing with logic?

Deductive reasoning and formal logic

Deductive reasoning argues from the general to a specific instance. The basic idea is that if something is true of a class of things in general, this truth applies to all legitimate members of that class.

All human beings are mortal. Socrates is human.
Therefore, Socrates is mortal.

Deductive reasoning and formal logic

Deductive reasoning argues from the general to a specific instance. The basic idea is that if something is true of a class of things in general, this truth applies to all legitimate members of that class.

All human beings are mortal. Socrates is human.
Therefore, Socrates is mortal.

Formal Logic is a formal version of human deductive logic. It provides a formal language with an unambiguous syntax and a precise meaning, and it provides rules for manipulating expressions in a way that respects this meaning.

$$\frac{\forall X.(\text{human}(X) \Rightarrow \text{mortal}(X)) \qquad \text{human}(\text{Socrates})}{\text{mortal}(\text{Socrates})}$$

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible.

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible. **Computational logic** is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and completeness of reasoning. However, it is also concerned with efficiency.

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible. **Computational logic** is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and completeness of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible. **Computational logic** is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and completeness of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible. **Computational logic** is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and completeness of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
?- yes
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible. **Computational logic** is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and completeness of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
?- yes

?- mortal(Y).
```

Computational logic

The existence of a formal language for representing information and the existence of a corresponding set of mechanical manipulation rules together make **automated reasoning using computers** possible. **Computational logic** is a branch of mathematics that is concerned with the theoretical underpinnings of automated reasoning. Like Formal Logic, Computational Logic is concerned with precise syntax and semantics and correctness and completeness of reasoning. However, it is also concerned with efficiency.

```
mortal(X) :- human(X).
human(socrates).

?- mortal(socrates).
?- yes

?- mortal(Y).
?- Y = socrates ? ;
no
```

Outline

Intentional Systems

When explaining human activity, it is often useful to make statements such as the following:

- Janine took her umbrella because she *believed* it was going to rain.
- Michael worked hard because he *wanted* to possess a PhD.

Intentional Systems

When explaining human activity, it is often useful to make statements such as the following:

- Janine took her umbrella because she *believed* it was going to rain.
- Michael worked hard because he *wanted* to possess a PhD.

These statements make use of a “folk psychology”, by which human behaviour is predicted and explained through the attribution of attitudes, such as believing, wanting, hoping, fearing,...

Intentional Systems

When explaining human activity, it is often useful to make statements such as the following:

- Janine took her umbrella because she *believed* it was going to rain.
- Michael worked hard because he *wanted* to possess a PhD.

These statements make use of a “folk psychology”, by which human behaviour is predicted and explained through the attribution of attitudes, such as believing, wanting, hoping, fearing,...

The attitudes employed in such folk psychological descriptions are called the *intentional notions*.

Intentional Systems

When explaining human activity, it is often useful to make statements such as the following:

- Janine took her umbrella because she *believed* it was going to rain.
- Michael worked hard because he *wanted* to possess a PhD.

These statements make use of a “folk psychology”, by which human behaviour is predicted and explained through the attribution of attitudes, such as believing, wanting, hoping, fearing,...

The attitudes employed in such folk psychological descriptions are called the *intentional notions*.

Intentional system = system made up of entities whose behaviour can be predicted by the method of attributing belief, desires and rational acumen.

Intentional Systems

When explaining human activity, it is often useful to make statements such as the following:

- Janine took her umbrella because she *believed* it was going to rain.
- Michael worked hard because he *wanted* to possess a PhD.

These statements make use of a “folk psychology”, by which human behaviour is predicted and explained through the attribution of attitudes, such as believing, wanting, hoping, fearing,...

The attitudes employed in such folk psychological descriptions are called the *intentional notions*.

Intentional system = system made up of entities whose behaviour can be predicted by the method of attributing belief, desires and rational acumen.

[D. C. Dennett, The Intentional Stance, 1989]

Agents, strong and weak definitions

An agent is an hardware or software system

- situated
- autonomous
- flexible
    - reactive
    - proactive
    - social

N. Jennings, K. Sycara, M. Wooldridge, A Roadmap of Agent Research and Development, JAAMAS 1(1), 1998

Agents, strong and weak definitions

An agent is an hardware or software system

- situated
- autonomous
- flexible
  - reactive
  - proactive
  - social

N. Jennings, K. Sycara, M. Wooldridge, A Roadmap of Agent Research and Development, JAAMAS 1(1), 1998

Besides being characterised by the notions identified by N. Jennings, K. Sycara, M. Wooldridge ("weak" definition), an agent may be conceptualised following an anthropomorphic approach ("strong" definition).

Agents, strong and weak definitions

An agent is an hardware or software system

- situated
- autonomous
- flexible
  - reactive
  - proactive
  - social

N. Jennings, K. Sycara, M. Wooldridge, A Roadmap of Agent Research and Development, JAAMAS 1(1), 1998

Besides being characterised by the notions identified by N. Jennings, K. Sycara, M. Wooldridge ("weak" definition), an agent may be conceptualised following an anthropomorphic approach ("strong" definition).

Y. Shoham, Agent-oriented programming, Artificial Intelligence, 60(1), 1993; A. S. Rao, M. P. Georgeff, An abstract architecture for rational agents, KR&R-92

Agents as Intentional Systems

- If we adhere to the “strong definition” of agents, intentional systems are a suitable theory for agents.

Agents as Intentional Systems

- If we adhere to the “strong definition” of agents, intentional systems are a suitable theory for agents.
- The representation of intentional notions raises a set of delicate technical questions, both on the syntactic and the semantic side.

Agents as Intentional Systems

- If we adhere to the “strong definition” of agents, intentional systems are a suitable theory for agents.
- The representation of intentional notions raises a set of delicate technical questions, both on the syntactic and the semantic side.
- Modal logic languages are suitable for **specifying** agents as intentional systems.

Agents as Intentional Systems

- If we adhere to the “strong definition” of agents, intentional systems are a suitable theory for agents.
- The representation of intentional notions raises a set of delicate technical questions, both on the syntactic and the semantic side.
- Modal logic languages are suitable for **specifying** agents as intentional systems.
- Languages based on computational (modal) logic are suitable for **programming agents**.

Agents as Intentional Systems

- If we adhere to the “strong definition” of agents, intentional systems are a suitable theory for agents.
- The representation of intentional notions raises a set of delicate technical questions, both on the syntactic and the semantic side.
- Modal logic languages are suitable for **specifying** agents as intentional systems.
- Languages based on computational (modal) logic are suitable for **programming agents**.
- Axiomatised logic-based languages can undergo an axiomatic **verification**; other languages which can be used for **model checking**.

Modal Logic

Modal logic is an extension of classical logic with (generally) a new connective $\Box$ and its derivable counterpart $\Diamond$, known as *necessity* and *possibility* respectively.

Modal Logic

Modal logic is an extension of classical logic with (generally) a new connective $\Box$ and its derivable counterpart $\Diamond$, known as *necessity* and *possibility* respectively.

If a formula $\Box p$ is true, it means that p is necessarily true, i.e. true in every possible scenario, and $\Diamond p$ means that p is possibly true, i.e. true in at least one possible scenario.

Modal Logic

Modal logic is an extension of classical logic with (generally) a new connective $\Box$ and its derivable counterpart $\Diamond$, known as *necessity* and *possibility* respectively.

If a formula $\Box p$ is true, it means that p is necessarily true, i.e. true in every possible scenario, and $\Diamond p$ means that p is possibly true, i.e. true in at least one possible scenario.

It is possible to define $\Diamond$ in terms of $\Box$:

$$\Diamond p \Leftrightarrow \neg \Box \neg p$$

Modal Logic

Different kinds of modal logics exist:

- epistemic logic
- temporal logic
- deontic logic
- dynamic logic
- ... and combinations of them (BDI logic, KARO logic, ...)

Outline

AGENT-0

In the already cited paper "*Agent-oriented programming, Artificial Intelligence 60(1), 1993*", Shoham proposes that a fully developed AOP ("Agent-Oriented Programming") system will have three components:

1. a logical system for defining the mental state of agents;

AGENT-0

In the already cited paper "*Agent-oriented programming, Artificial Intelligence 60(1), 1993*", Shoham proposes that a fully developed AOP ("Agent-Oriented Programming") system will have three components:

1. a logical system for defining the mental state of agents;
2. an interpreted programming language for programming agents;

AGENT-0

In the already cited paper "*Agent-oriented programming, Artificial Intelligence 60(1), 1993*", Shoham proposes that a fully developed AOP ("Agent-Oriented Programming") system will have three components:

1. a logical system for defining the mental state of agents;
2. an interpreted programming language for programming agents;
3. an agentification process, for compiling agent programs into low-level executable systems.

AGENT-0

In the already cited paper "*Agent-oriented programming, Artificial Intelligence 60(1), 1993*", Shoham proposes that a fully developed AOP ("Agent-Oriented Programming") system will have three components:

1. a logical system for defining the mental state of agents;
2. an interpreted programming language for programming agents;
3. an agentification process, for compiling agent programs into low-level executable systems.

However, he only describes the first two components.

AGENT-0: the father

AGENT-0: the logic behind it

BDI logic, combination of:

- temporal logic (linear time in Cohen and Levesque, branching time in Rao and Georgeff)
- modal logic(s) of belief, desires & goals (intentions)

AGENT-0: the logic behind it

BDI logic, combination of:

- temporal logic (linear time in Cohen and Levesque, branching time in Rao and Georgeff)
- modal logic(s) of belief, desires & goals (intentions)

The modalities of Rao and Georgeff's BDI logic are BEL(ϕ), GOAL(ϕ), INTEND(ϕ).

AGENT-0: the logic behind it

BDI logic, combination of:

- temporal logic (linear time in Cohen and Levesque, branching time in Rao and Georgeff)
- modal logic(s) of belief, desires & goals (intentions)

The modalities of Rao and Georgeff's BDI logic are BEL(ϕ), GOAL(ϕ), INTEND(ϕ).

[P.R. Cohen and H.J. Levesque. Intention is choice with commitment. Artificial Intelligence, 1990]
[A. S. Rao and M. P. Georgeff. Decision Procedures for BDI Logics. Journal of Logic and Computation, 1998]

AGENT-0: the logic behind it

Which relationships among BDI modalities?
Some possible axioms...

AGENT-0: the logic behind it

Which relationships among BDI modalities?
Some possible axioms...

- INTEND(does(e)) $\Rightarrow$ does(e)
  (intention leading to action)

AGENT-0: the logic behind it

Which relationships among BDI modalities?
Some possible axioms...

- INTEND(does(e)) $\Rightarrow$ does(e)
  (intention leading to action)
- done(e) $\Rightarrow$ BEL(done(e))
  (awareness of primitive events)

AGENT-0: the logic behind it

Which relationships among BDI modalities?
Some possible axioms...

- INTEND(does(e)) $\Rightarrow$ does(e)
  (intention leading to action)
- done(e) $\Rightarrow$ BEL(done(e))
  (awareness of primitive events)
- INTEND(ϕ) $\Rightarrow$ inevitable $\Diamond(\neg$ INTEND(ϕ))
  (no infinite deferral)

AGENT-0: the syntax

```
<program>     ::= <timegrain> <fact>* <capability>* <commitrule>*
<timegrain>   ::= m | h | d | y
<capability>  ::= (<action> <mntlcond>)
<commitrule>  ::= (COMMIT <msgcond> <mntlcond>
                          (<agent> <action>)*)
<msgcond>     ::= <msgconj> | (<msgcond> OR <msgcond>)
<msgconj>     ::= <msgpattern> | (<msconj> AND <msconj>)
<msgpattern>  ::= (<agent> INFORM <fact>) |
                  (<agent> REQUEST <action>) |
                  (NOT <msgpattern>)
<mntlcond>    ::= <mntlconj> | (<mntlcond> OR <mntlcond>)
<mntlconj>    ::= <mntlpattern> | (<mntlconj> AND <mntlconj>)
<mntlpattern>::= (B <fact>) | ((CMT <agent>) <action>) |
                  (NOT <mntlpattern>)
```

AGENT-0: syntax

```
<action>      ::= (DO         <time> <privateaction>) |
                  (INFORM     <time> <agent> <fact>)  |
                  (REQUEST    <time> <agent> <action>)|
                  (UNREQUEST  <time> <agent> <action>)|
                  (REFRAIN    <action>)               |
                  (IF         <mntlcond> <action>)
<fact>        ::= (<time> <atom>) | (NOT (<time> <atom>))
<atom>        ::= atomi Prolog (predicati con argomenti)
<time>        ::= integer |now | <time> + <time>
<agent>       ::= <string> | <variable>
<variable>    ::= ?<string> | ?!<string>
```

AGENT-0: the syntax

An AGENT-0 program consists of a **knowledge base** made up of facts, a set of **capabilities** and a set of **commitment rules** (together with all the “bricks” for composing them).

AGENT-0: the syntax

An AGENT-0 program consists of a **knowledge base** made up of facts, a set of **capabilities** and a set of **commitment rules** (together with all the "bricks" for composing them).

Facts are atomic sentences of a simple temporal language: *(t atom)*, *(NOT (t atom))*.
Example: *(0 (stored orange 1000))*.

AGENT-0: the syntax

Capabilities have the form

(action mentalcondition)

meaning that the agent is capable of performing *action* if *mentalcondition* is true.

AGENT-0: the syntax

Capabilities have the form

(action mentalcondition)

meaning that the agent is capable of performing *action* if *mentalcondition* is true.

Commitment rules have the form

(COMMIT messagecondition mentalcondition (agent action))*

where *messagecondition* and *mentalcondition* are message and mental conditions, resp., *agent* is the name of the agent toward which the commitment is taken, *action* is an action and *** means “zero or more”.

AGENT-0: the syntax

```
(COMMIT  (?a REQUEST ?action)
         (B (now (myfriend ?a)))
         (?a ?action ))
```

AGENT-0: the semantics

No formal semantics for the language is given.

AGENT-0: the interpreter

The AGENT-0 engine is characterized by the following two-step cycle:

1. Read the current messages and update beliefs and commitments.
2. Execute the commitments for the current time, possibly resulting in further belief change.

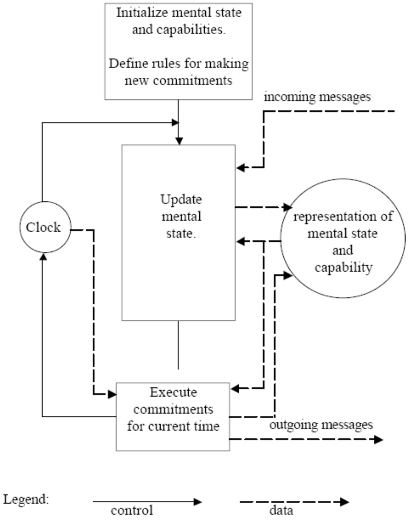
AGENT-0: the interpreter

The AGENT-0 engine is characterized by the following two-step cycle:

1. Read the current messages and update beliefs and commitments.
2. Execute the commitments for the current time, possibly resulting in further belief change.

Actions to which agents can be committed include communicative ones such as informing and requesting, as well as arbitrary private actions.

AGENT-0: the interpreter



AGENT-0: the implementations

A prototype AGENT-0 interpreter has been implemented in Common Lisp and has been installed on Sun/Unix, DecStation/Ultrix and Macintosh computers.

AGENT-0: the implementations

A prototype AGENT-0 interpreter has been implemented in Common Lisp and has been installed on Sun/Unix, DecStation/Ultrix and Macintosh computers.

A separate implementation has been developed by Hewlett Packard as part of a joint project to incorporate AOP in the New Wave^{TM} architecture.

AGENT-0: the extensions

Two extensions of AGENT-0 have been proposed:

AGENT-0: the extensions

Two extensions of AGENT-0 have been proposed:

- PLACA enriches AGENT-0 with a mechanism for flexible management of plans.

AGENT-0: the extensions

Two extensions of AGENT-0 have been proposed:

- PLACA enriches AGENT-0 with a mechanism for flexible management of plans.
- Agent-K is an attempt to standardize the message passing functionality in AGENT-0. It combines the syntax of AGENT-0 (without support for the planning mechanisms of PLACA) with the format of KQML (*Knowledge Query and Manipulation Language*) to ensure that messages written in languages different from AGENT-0 can be handled.

AGENT-0: the extensions

Two extensions of AGENT-0 have been proposed:

- PLACA enriches AGENT-0 with a mechanism for flexible management of plans.
- Agent-K is an attempt to standardize the message passing functionality in AGENT-0. It combines the syntax of AGENT-0 (without support for the planning mechanisms of PLACA) with the format of KQML (*Knowledge Query and Manipulation Language*) to ensure that messages written in languages different from AGENT-0 can be handled.

[Thomas, S. R. The PLACA agent programming language. In ATAL'94]
[Davies, W. H. and Edwards, P. Agent-K: An integration of AOP & KQML. Workshop on Intelligent Information Agents associated with CIKM'94]

AGENT-0: the applications

AGENT-0 is suitable for modeling agents and MAS.

AGENT-0: the applications

AGENT-0 is suitable for modeling agents and MAS.

We are not aware of documents showing the suitability of AGENT-0 or its extensions for verifying MAS specifications or implementing real agent systems.

Outline

AgentSpeak(L)

- AgentSpeak(L) [A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language, MAAMAW'96] takes as its starting point the "procedural reasoning system" PRS and its dMARS implementation.

AgentSpeak(L)

- AgentSpeak(L) [A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language, MAAMAW'96] takes as its starting point the "procedural reasoning system" PRS and its dMARS implementation.
- AgentSpeak(L) is based on a restricted first-order language with events and actions.

AgentSpeak(L)

- AgentSpeak(L) [A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language, MAAMAW'96] takes as its starting point the “procedural reasoning system” PRS and its dMARS implementation.
- AgentSpeak(L) is based on a restricted first-order language with events and actions.
- Beliefs, desires and intentions of the agent are not represented as modal formulas, but they are ascribed to agents, in an implicit way, at design time.

AgentSpeak(L)

- AgentSpeak(L) [A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language, MAAMAW'96] takes as its starting point the "procedural reasoning system" PRS and its dMARS implementation.
- AgentSpeak(L) is based on a restricted first-order language with events and actions.
- Beliefs, desires and intentions of the agent are not represented as modal formulas, but they are ascribed to agents, in an implicit way, at design time.
- The current state of the agent can be viewed as its current belief base; states that the agent wants to bring about can be viewed as desires; and the adoption of programs to satisfy such stimuli can be viewed as intentions.

AgentSpeak(L): the fathers

AgentSpeak(L): the logic behind it

Rao and Georgeff's BDI logic.

AgentSpeak(L): the syntax

$$
\begin{array}{llll}
ag & ::= & bs \quad ps & \\
bs & ::= & at_1 \ldots at_n & (n \geq 0) \\
at & ::= & P(t_1, \ldots t_n) & (n \geq 0) \\
ps & ::= & p_1 \ldots p_n & (n \geq 1) \\
p & ::= & te : ct \leftarrow h & \\
te & ::= & +at \mid -at \mid +g \mid -g & \\
ct & ::= & at \mid \neg at \mid ct \ \wedge \ ct \mid \mathsf{T} & \\
h & ::= & a \mid g \mid u \mid h; h & \\
g & ::= & !at \mid \ ?at & \\
u & ::= & +at \mid -at &
\end{array}
$$

AgentSpeak(L): the syntax

Initial beliefs

```
adjacent(a,b).
adjacent(b,c).
adjacent(c,d).
location(robot,a).
location(waste,b).
location(bin,d).
```

AgentSpeak(L): the syntax

Plans

```
+location(waste,X):location(robot,X) &
                   location(bin,Y)
                   <- pick(waste);
                      !location(robot,Y);
                      drop(waste).              (P1)
```

AgentSpeak(L): the syntax

```
+!location(robot,X):location(robot,X) <- true.   (P2)
```

AgentSpeak(L): the syntax

```
+!location(robot,X):location(robot,Y) &
                    (not (X = Y)) &
                          adjacent(Y,Z) &
                          (not (location(car, Z)))
                                <- move(Y,Z);
                                   +!location(robot,X). (P3)
```

AgentSpeak(L): the syntax

Triggering event

```
+!location(robot,b).
```

AgentSpeak(L): the syntax

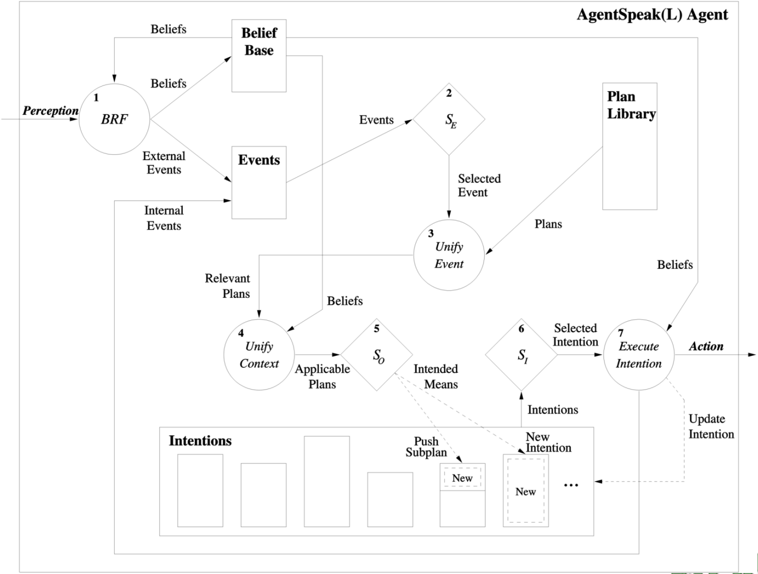
Intention (instantiated plan)

```
[+!location(robot,b): location(robot,a) &
                       not(b = a) &
                       adjacent(a, b) &
                       not(location(car,b)) <-
                            move(a,b);
                            +!location(robot,b)].
```

AgentSpeak(L): the semantics

AgentSpeak(L) has a formal operational semantics and a proof theory based on labeled transition systems.

AgentSpeak(L): the interpreter



AgentSpeak(L): the implementations

Prototypical interpreters for BDI-like languages and for AgentSpeak(L) in particular have been developed in the past.

The Jadex reasoning engine follows the BDI model and facilitates easy intelligent agent construction with sound software engineering foundations.

It allows for programming intelligent software agents in XML and Java and can be deployed on different kinds of middleware such as JADE.

Jadex is available open source at `http://jadex.informatik.uni-hamburg.de`.

AgentSpeak(L): the implementations

A new interpreter and multi-agent platform for AgentSpeak(L) called Jason has been recently developed.

The interpreter implements, in Java, the operational semantics of an extended version of AgentSpeak(L).

Jason is available open source at `http://jason.sourceforge.net`.

AgentSpeak(L): the extensions

The community working on AgentSpeak(L) is, and has been in the past, very active. Thus, many extensions to AgentSpeak(L) exist.

- Cooperation through plan exchange [D. Ancona, V. Mascardi, J. Hübner and R. Bordini. Coo-AgentSpeak: Cooperation in AgentSpeak through Plan Exchange. AAMAS 2004].
- Ontological reasoning [A. F. Moreira, R. Vieira, R. H. Bordini, and J. F. Hübner. Agent-oriented programming with underlying ontological reasoning. DALT III. 2005]
- Belief revision [N. Alechina, R. H. Bordini, J. F. Hübner, M. Jago, B. Logan. Belief revision for AgentSpeak agents. AAMAS 2006]
- Team formation [J. F. Hübner, R. H. Bordini. Developing a Team of Gold Miners Using Jason. PROMAS 2007]
- Semantic Web [T. Klapiscak, R. H. Bordini. JASDL: A Practical Programming Approach Combining Agent and Semantic Web Technologies. DALT 2008]
- ...

AgentSpeak(L): the extensions

Jason Home Description Documents Examples Demos Related Projects

These are some of the project contributing, extending, or using *Jason*:

- J-Moise allows ***Jason*** agents to use Moise+ organisations (available with the ***Jason*** distribution).
- There is an interface between ***Jason*** and Cartago, available here.
- JASDL is a programming approach combining agent-oriented programming and semantic web technologies, developed by Tom Klapiscak (to be released open source soon).
- There is a ***Jason*** plug-in for Eclipse, developed by Germano Fronza.
- Some really brave folks are porting the whole of ***Jason*** to C/C++! See here.
- The MADeM project on multi-modal decision making is using ***Jason***, see this paper.
  NEWS: J-MADeM is now available for download in the tools area of the ***Jason*** download page.
- The Talos project (on self-organisation) also uses ***Jason***.
- There are various projects extending ***Jason*** here.
- CASO is a project extending AgentSpeak with constraint solving.
- TankCoders is a 3D environment where agents are run using ***Jason***.
- The Extrospective Agent project uses Cartago and ***Jason***.

AgentSpeak(L): the applications

Formal verification
In a series of papers, Bordini et al. have developed model-checking techniques that apply directly to multi-agent programs written in AgentSpeak(L).
The approach is to translate AgentSpeak(L) multi-agent systems into either Promela or Java models, then using, respectively, SPIN or JPF as model checkers.

AgentSpeak(L): the applications

Formal verification

In a series of papers, Bordini et al. have developed model-checking techniques that apply directly to multi-agent programs written in AgentSpeak(L).
The approach is to translate AgentSpeak(L) multi-agent systems into either Promela or Java models, then using, respectively, SPIN or JPF as model checkers.

[R. H. Bordini, M. Fisher, W. Visser, M. Wooldridge. Verifying multi-agent programs by model checking. JAAMAS (2):239-256. 2006.]

AgentSpeak(L): the applications

Implementation of “real” agent systems

AgentSpeak(L): the applications

Implementation of “real” agent systems

No “witnessed” applications.

AgentSpeak(L): the applications

Implementation of "real" agent systems

No "witnessed" applications.

The Jason implementation of AgentSpeak(L) raised the interest of companies in France, the US, and Germany: Jason's developers received technical questions from people working there. Difficult to know if these companies actually made a choice to use Jason in the end.

AgentSpeak(L): the applications

Implementation of "real" agent systems

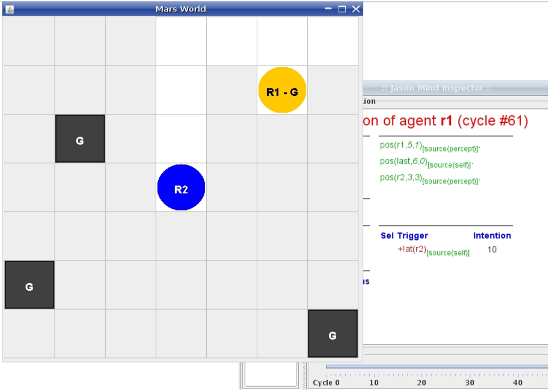
No "witnessed" applications.

The Jason implementation of AgentSpeak(L) raised the interest of companies in France, the US, and Germany: Jason's developers received technical questions from people working there. Difficult to know if these companies actually made a choice to use Jason in the end.

In the past various systems have been developed using ad hoc implementations of PRS or reactive planning systems more generally: air traffic control, control of an oil processing plant, ...

Jason: a Short Demo

```
// mars robot 1

/* Initial beliefs */

at(P) :- pos(P,X,Y) & pos(r1,X,Y).

/* Initial goal */

!check(slots).

/* Plans */

+!check(slots) : not garbage(r1)
   <- next(slot);
      !!check(slots).
+!check(slots).


+garbage(r1) : not .desire(carry_to(r2))
   <- !carry_to(r2).

+!carry_to(R)
   <- // remember where to go back
      ?pos(r1,X,Y);
      -+pos(last,X,Y);

      // carry garbage to r2
      !take(garb,R);

      // goes back and continue to check
      !at(last);
      !!check(slots).

+!take(S,L) : true
   <- !ensure_pick(S);
      !at(L);
      drop(S).
```



Outline

Concurrent METATEM

M. Fisher and H. Barringer. Concurrent METATEM Processes – A Language for Distributed AI, in Proc. of the European Simulation Multiconference, 1991

Concurrent METATEM is a language based upon the direct execution of temporal formulae.

It consists of two distinct aspects:

1. an execution mechanism for temporal formulae in a particular form; and
2. an operational model that treats single executable temporal logic programs as asynchronously executing agents in a concurrent agent-based system.

Concurrent METATEM: the father

Concurrent METATEM: the logic behind it

FML is a first-order temporal logic based on discrete, linear models with finite past and infinite future.

Concurrent METATEM: the logic behind it

FML is a first-order temporal logic based on discrete, linear models with finite past and infinite future.

FML introduces two new connectives to classical logic, *until* ($\mathcal{U}$) and *since* ($\mathcal{S}$), together with other operators definable in terms of $\mathcal{U}$ and $\mathcal{S}$.

Concurrent METATEM: the logic behind it

FML is a first-order temporal logic based on discrete, linear models with finite past and infinite future.

FML introduces two new connectives to classical logic, *until* ($\mathcal{U}$) and *since* ($\mathcal{S}$), together with other operators definable in terms of $\mathcal{U}$ and $\mathcal{S}$.

The intuitive meaning of a temporal logic formula

$$\varphi\,\mathcal{U}\psi$$

is that ψ will become true at some future time point *t* and that in all states between and different from now and *t*, φ will be true. $\mathcal{S}$ is the analogous of $\mathcal{U}$ in the past.

Concurrent METATEM: the logic behind it

FML is a first-order temporal logic based on discrete, linear models with finite past and infinite future.

FML introduces two new connectives to classical logic, *until* ($\mathcal{U}$) and *since* ($\mathcal{S}$), together with other operators definable in terms of $\mathcal{U}$ and $\mathcal{S}$.

The intuitive meaning of a temporal logic formula

$$\varphi\,\mathcal{U}\psi$$

is that ψ will become true at some future time point *t* and that in all states between and different from now and *t*, φ will be true. $\mathcal{S}$ is the analogous of $\mathcal{U}$ in the past.

[Fisher, M. A normal form for first-order temporal formulae. CADE'92]

Concurrent METATEM: the syntax

$\psi\,\mathcal{U}\phi$	: ψ will be true until ϕ will become true	primitive
$\psi\,\mathcal{S}\phi$	: ψ was true until ϕ became true	primitive
$\bigcirc\phi$	: ϕ is true in the next state	[**false** $\mathcal{U}\phi$]
$\circledcirc\phi$	: there was a last state and ϕ was true in it	[**false** $\mathcal{S}\phi$]
$\odot\phi$	: if there was a last state, ϕ was true in it	[$\neg \circledcirc \neg\phi$]
$\Diamond\phi$	: ϕ will be true in some future state	[**true** $\mathcal{U}\phi$]
$\blacklozenge\phi$	: ϕ was true in some past state	[**true** $\mathcal{S}\phi$]
$\Box\phi$	: ϕ will be true in all future states	[$\neg \Diamond\neg\phi$]
$\blacksquare\phi$	: ϕ was true in all past states	[$\neg\blacklozenge\neg\phi$]

Concurrent METATEM: the semantics

METATEM semantics is the one defined for FML.

Concurrent METATEM: the interpreter

The computational engine of an object is based on the METATEM paradigm of executable temporal logics.

Concurrent METATEM: the interpreter

The computational engine of an object is based on the METATEM paradigm of executable temporal logics.

The idea behind this approach is to directly execute a declarative agent specification given as a set of program rules which are temporal logic formulae of the form:

$$\text{antecedent about past} \Rightarrow \text{consequent about future}$$

Concurrent METATEM: the interpreter

The computational engine of an object is based on the METATEM paradigm of executable temporal logics.

The idea behind this approach is to directly execute a declarative agent specification given as a set of program rules which are temporal logic formulae of the form:

$$\text{antecedent about past} \Rightarrow \text{consequent about future}$$

The past-time antecedent is a temporal logic formula referring strictly to the past, whereas the future time consequent is a temporal logic formula referring either to the present or future. The intuitive interpretation of such a rule is *"on the basis of the past, do the future".*

Concurrent METATEM: the implementations

Two implementations of the imperative future paradigm upon which Concurrent METATEM is based exist.

1. The first is a prototype interpreter for propositional METATEM implemented in the Scheme language [M. Fisher. Implementing a prototype METATEM interpreter. Tech. rep., Department of Computer Science, University of Manchester. 1990].

Concurrent METATEM: the implementations

Two implementations of the imperative future paradigm upon which Concurrent METATEM is based exist.

1. The first is a prototype interpreter for propositional METATEM implemented in the Scheme language [M. Fisher. Implementing a prototype METATEM interpreter. Tech. rep., Department of Computer Science, University of Manchester. 1990].
2. A more robust Prolog-based interpreter for a restricted first-order version of METATEM has been used as a transaction programming language for temporal databases [M. Finger, M. Fisher, R. Owens. METATEM at work: Modelling reactive systems using executable temporal logic. IEA/AIE'93].

Concurrent METATEM: the extensions

- Single Concurrent METATEM agents have been extended with deliberation and beliefs [M. Fisher. Implementing BDI-like systems by direct execution. IJCAI'97] and with resource-bounded reasoning [M. Fisher, C. Ghidini. Programming resource-bounded deliberative agents. IJCAI'99].
- Compilation techniques for MASs specified in Concurrent METATEM are analyzed in [A. Kellet and M. Fisher. Automata representations for concurrent METATEM. TIME'97].
- Concurrent METATEM has been proposed as a coordination language in [A. Kellet and M. Fisher. Concurrent METATEM as a coordination language. COORDINATION'97].
- The definition of groups of agents in Concurrent METATEM is discussed in [M. Fisher. Representing abstract agent architectures. ATAL'98; M. Fisher and T. Kakoudakis. Flexible agent grouping in executable temporal logic. ISPLIP'99]
- Confidence is added to both single and multiple agents in [M Fisher and C. Ghidini. The ABC of rational agent programming. AAMAS'02].
- The development of teams of agents is discussed in [B. Hirsch, M. Fisher, C. Ghidini. Organising logic-based agents. FAABS II, 2002].

Concurrent METATEM: the applications

In [M. Fisher. A survey of Concurrent METATEM – the language and its applications. ICTL'94] a range of sample applications of Concurrent METATEM utilizing both the core features of the language and some of its extensions are discussed.
They include bidding, problem solving, process control, fault tolerance.

Concurrent METATEM: the applications

In [M. Fisher. A survey of Concurrent METATEM – the language and its applications. ICTL'94] a range of sample applications of Concurrent METATEM utilizing both the core features of the language and some of its extensions are discussed.
They include bidding, problem solving, process control, fault tolerance.

Concurrent METATEM has the potential of **specifying and verifying** applications in all of the areas above [M. Fisher, M. Wooldridge. On the formal specification and verification of multi-agent systems. International Journal of Cooperative Information Systems, 1997], but **we are not aware of the development of real systems using Concurrent METATEM**.

Outline

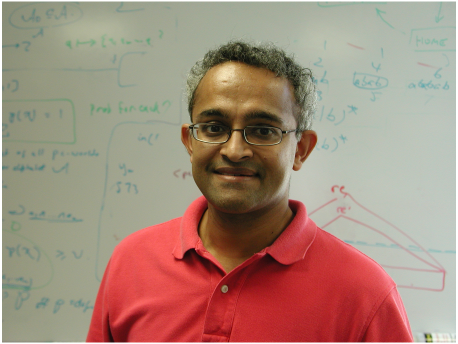
IMPACT

Interactive Maryland Platform for Agents Collaborating Together

THE IMPACT PROJECT

IMPACT is an international research project led by the University of Maryland. The principal **goal** of the IMPACT project is to develop both a theory as well as a software implementation that facilitates the **creation, deployment, interaction, and collaborative aspects of software agents in a heterogeneous, distributed environment.** IMPACT provides a set of servers (yellow pages, thesaurus, registration, type and interface) that facilitate agent interoperability in an application independent manner. It also provides an Agent Development Environment for creating, testing, and deploying agents.

IMPACT: the father



IMPACT: the logic behind it

Deontic logic is the logic to reason about ideal and actual behavior.

IMPACT: the logic behind it

Deontic logic is the logic to reason about ideal and actual behavior.

From the 1950s, von Wright, Castañeda, Alchourrón and Bulygin and others developed deontic logic as a modal logic with operators for permission, obligation and prohibition.

IMPACT: the logic behind it

Deontic logic is the logic to reason about ideal and actual behavior.

From the 1950s, von Wright, Castañeda, Alchourrón and Bulygin and others developed deontic logic as a modal logic with operators for permission, obligation and prohibition.

Deontic logic has traditionally been used to analyze the structure of normative law and normative reasoning in law.

IMPACT: the logic behind it

Deontic logic is the logic to reason about ideal and actual behavior.

From the 1950s, von Wright, Castañeda, Alchourrón and Bulygin and others developed deontic logic as a modal logic with operators for permission, obligation and prohibition.

Deontic logic has traditionally been used to analyze the structure of normative law and normative reasoning in law.

[G. H. von Wright. Deontic logic. Mind 60, 1951]
[C. E. Alchourrón, E. Bulygin, Normative Systems. 1971.]
[N.-N. Castañeda. Thinking and Doing. The Philosophical Foundations of Institutions. 1975.]

IMPACT: the syntax

Very complex since it includes “code calls” towards external pieces of software and integrity constraints.

IMPACT: the syntax

Very complex since it includes "code calls" towards external pieces of software and integrity constraints.

The basic idea is that each agent has a set of rules specifying the principles under which the agent is operating. These rules specify, using deontic modalities, what the agent may do, must do, may not do, etc. and may include conditions over "code calls".

IMPACT: the semantics

If an agent's behavior is defined by a program $\mathcal{P}$, the question that the agent must answer, over and over again is:

> What is the set of all action status atoms of the form **Do** $\alpha(\vec{t})$ that are true with respect to $\mathcal{P}$, the current state $\mathcal{O}$ and the set $\mathcal{I}C$ of underlying integrity constraints on agent states?

IMPACT: the semantics

If an agent's behavior is defined by a program $\mathcal{P}$, the question that the agent must answer, over and over again is:

> What is the set of all action status atoms of the form **Do** $\alpha(\vec{t})$ that are true with respect to $\mathcal{P}$, the current state $\mathcal{O}$ and the set $\mathcal{I}C$ of underlying integrity constraints on agent states?

This set defines the actions the agent must take; [T. Eiter, V.S. Subrahmanian, G. Pick. Heterogeneous active agents, I: Semantics. Artificial Intelligence. 1999] provides a series of successively more refined semantics for action programs that answer this question.

IMPACT: the implementations

The implementation of the IMPACT agent program consists of two major parts, both implemented in Java:

1. the IMPACT Agent Development Environment (IADE) which is used by the developer to build and compile agents, and
2. the run-time part that allows the agent to autonomously update its reasonable status set and execute actions as its state changes.

IMPACT: the implementations

The implementation of the IMPACT agent program consists of two major parts, both implemented in Java:

1. the IMPACT Agent Development Environment (IADE) which is used by the developer to build and compile agents, and
2. the run-time part that allows the agent to autonomously update its reasonable status set and execute actions as its state changes.

The IADE provides a network accessible interface through which an agent developer can specify the data types, functions, actions, integrity constraints, notion of concurrency and agent program associated with her/his agent; it also provides support for compilation and testing.

IMPACT: the implementations

The implementation of the IMPACT agent program consists of two major parts, both implemented in Java:

1. the IMPACT Agent Development Environment (IADE) which is used by the developer to build and compile agents, and
2. the run-time part that allows the agent to autonomously update its reasonable status set and execute actions as its state changes.

The IADE provides a network accessible interface through which an agent developer can specify the data types, functions, actions, integrity constraints, notion of concurrency and agent program associated with her/his agent; it also provides support for compilation and testing.

The runtime execution module runs as a background applet and performs the following steps: (i) monitoring of the agent's message box, (ii) execution of the algorithm for updating the reasonable status set and (iii) execution of the actions α such that **Do**α is in the updated reasonable status set.

IMPACT: the extensions

Many extensions to the IMPACT framework are discussed in the book [V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, R. Ross. Heterogenous Active Agents. 2000] which analyses:

- *meta agent programs* to reason about other agents based on the beliefs they hold;
- *temporal agent programs* to specify temporal aspects of actions and states;
- *probabilistic agent programs* to deal with uncertainty; and
- *secure agent programs* to provide agents with security mechanisms.

IMPACT: the extensions

Many extensions to the IMPACT framework are discussed in the book [V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, R. Ross. Heterogenous Active Agents. 2000] which analyses:

- *meta agent programs* to reason about other agents based on the beliefs they hold;
- *temporal agent programs* to specify temporal aspects of actions and states;
- *probabilistic agent programs* to deal with uncertainty; and
- *secure agent programs* to provide agents with security mechanisms.

Agents able to recover from an integrity constraints violation and able to continue to process some requests while continuing to recover are discussed in [T, Eiter, V. Mascardi, V. S. Subrahmanian. Error-Tolerant Agents. Computational Logic: Logic Programming and Beyond. 2002].

IMPACT: the extensions

Many extensions to the IMPACT framework are discussed in the book [V.S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Özcan, R. Ross. Heterogenous Active Agents. 2000] which analyses:

- *meta agent programs* to reason about other agents based on the beliefs they hold;
- *temporal agent programs* to specify temporal aspects of actions and states;
- *probabilistic agent programs* to deal with uncertainty; and
- *secure agent programs* to provide agents with security mechanisms.

Agents able to recover from an integrity constraints violation and able to continue to process some requests while continuing to recover are discussed in [T, Eiter, V. Mascardi, V. S. Subrahmanian. Error-Tolerant Agents. Computational Logic: Logic Programming and Beyond. 2002].

The integration of planning algorithms in the IMPACT framework is discussed in [J. Dix, H. Munoz-Avila, D. Nau. IMPACTing SHOP: Putting an AI planner into a Multi-Agent Environment. Annals of Mathematics and AI. 2003].

IMPACT: the applications

IMPACT's main purpose is to allow the integration of heterogeneous information sources and software packages.

IMPACT: the applications

IMPACT's main purpose is to allow the integration of heterogeneous information sources and software packages.

It has been used to develop real applications, mainly in collaboration with the US Military Academy, ranging from combat information management where IMPACT was used to provide yellow pages matchmaking services to aerospace applications where IMPACT technology has led to the development of a multiagent solution to the "controlled flight into terrain" problem.

IMPACT: the applications

IMPACT's main purpose is to allow the integration of heterogeneous information sources and software packages.

It has been used to develop real applications, mainly in collaboration with the US Military Academy, ranging from combat information management where IMPACT was used to provide yellow pages matchmaking services to aerospace applications where IMPACT technology has led to the development of a multiagent solution to the "controlled flight into terrain" problem.

The IADE environment provides support for monitoring the MAS evolution.

Related work

Other well-known and relevant agent programming languages based on computational logic exist:

Related work

Other well-known and relevant agent programming languages based on computational logic exist:

- The one originally named **DyLog** [M. Baldoni, L. Giordano, A. Martelli, V. Patti. Modeling agents in a logic action language. Workshop on Practical Reasoning Agents, associated with FAPR'00. 2000].

Related work

Other well-known and relevant agent programming languages based on computational logic exist:

- The one originally named **DyLog** [M. Baldoni, L. Giordano, A. Martelli, V. Patti. Modeling agents in a logic action language. Workshop on Practical Reasoning Agents, associated with FAPR'00. 2000].
- **DALI** [S. Costantini, A. Tocchio. The DALI Logic Programming Agent-Oriented Language. JELIA 2004]

Related work

- The languages defined as a part of the **SOCS** European Project [http://lia.deis.unibo.it/research/socs]

Related work

- The languages defined as a part of the **SOCS** European Project [http://lia.deis.unibo.it/research/socs]
- **Congolog** [G. De Giacomo, Y. Lespérance, H. J. Levesque. Congolog, a concurrent programming language based on the situation calculus. Artificial Intelligence 121, 109–169. 2000].

Related work

- The languages defined as a part of the **SOCS** European Project [http://lia.deis.unibo.it/research/socs]
- **Congolog** [G. De Giacomo, Y. Lespérance, H. J. Levesque. Congolog, a concurrent programming language based on the situation calculus. Artificial Intelligence 121, 109–169. 2000].
- **3APL** [K. Hindriks, F. De Boer, W. Van Der Hoek, J.-J. Meyer. Formal semantics for an abstract agent programming language. Intelligent Agents IV. 1998] and its related languages, **Dribble** [B. Van Riemsdijk, W. Van Der Hoek, J.-J. Meyer. Agent programming In Dribble: from beliefs to goals with plans. AAMAS 2003] and **Goal** [K. Hindriks, F. De Boer, W. Van Der Hoek, J.-J. Meyer. Agent programming with declarative goals. Intelligent Agents VII. 2001].

Conclusions

Some recent applications of agents and MASs based on computational logic (or, at least, on declarative approaches) exist.

Conclusions

Some recent applications of agents and MASs based on computational logic (or, at least, on declarative approaches) exist.

[G. S. Semmel, S. R. Davis, K. W. Leucht, D. A. Rowe, K. E. Smith, L. Boloni. Space Shuttle Ground Processing with Monitoring Agents. IEEE Intelligent Systems. 2006]

[Go4Flex `http://jadex.informatik.uni-hamburg.de/bin/view/Usages/Projects`]

[V. Mascardi, D. Briola, M. Martelli, R. Caccia, C. Milani. Monitoring and Diagnosing Railway Signalling with Logic-Based Distributed Agents. CISIS 2008]

Conclusions

Some recent applications of agents and MASs based on computational logic (or, at least, on declarative approaches) exist.

[G. S. Semmel, S. R. Davis, K. W. Leucht, D. A. Rowe, K. E. Smith, L. Boloni. Space Shuttle Ground Processing with Monitoring Agents. IEEE Intelligent Systems. 2006]
[Go4Flex `http://jadex.informatik.uni-hamburg.de/bin/view/Usages/Projects`]
[V. Mascardi, D. Briola, M. Martelli, R. Caccia, C. Milani. Monitoring and Diagnosing Railway Signalling with Logic-Based Distributed Agents. CISIS 2008]

However...

Conclusions

It is true that logical approaches to multi-agent systems are not widely used in the market (yet). However, we witness a growing interest of the stakeholders in technologies such as autonomic computing, service oriented architectures, mobile robotics, and e-trade.

Conclusions

It is true that logical approaches to multi-agent systems are not widely used in the market (yet). However, we witness a growing interest of the stakeholders in technologies such as autonomic computing, service oriented architectures, mobile robotics, and e-trade.

These are all domains very much related to the research being carried out in the MAS community. As concerns increase over the reliability and security of such systems and over the public's trust in these systems, so the use of logical approaches is likely to increase.

Conclusions

It is true that logical approaches to multi-agent systems are not widely used in the market (yet). However, we witness a growing interest of the stakeholders in technologies such as autonomic computing, service oriented architectures, mobile robotics, and e-trade.

These are all domains very much related to the research being carried out in the MAS community. As concerns increase over the reliability and security of such systems and over the public's trust in these systems, so the use of logical approaches is likely to increase.

Moreover, domains such as Semantic Web services have a huge market potential and enjoy extensive input from (specifically) computational logic-based agent systems research.

Conclusions

It is true that logical approaches to multi-agent systems are not widely used in the market (yet). However, we witness a growing interest of the stakeholders in technologies such as autonomic computing, service oriented architectures, mobile robotics, and e-trade.

These are all domains very much related to the research being carried out in the MAS community. As concerns increase over the reliability and security of such systems and over the public's trust in these systems, so the use of logical approaches is likely to increase.

Moreover, domains such as Semantic Web services have a huge market potential and enjoy extensive input from (specifically) computational logic-based agent systems research.

[M. Fisher, R. H. Bordini, B. Hirsch, P. Torroni. Computational Logics And Agents: A Road Map Of Current Technologies And Future Trends. Computational Intelligence, 23(1). 2007]

Sources of information

The content of this presentation is mainly based on

[V. Mascardi, M. Martelli, L. Sterling. Logic-Based Specification Languages for Intelligent Software Agents. Theory and Practice of Logic Programming Journal (TPLP), 4(4), Cambridge University Press, pagg. 429 – 494, 2004
`http://www.disi.unige.it/person/MascardiV/Papers/VivianaPublications.html`]

Sources of information

The content of this presentation is mainly based on

[V. Mascardi, M. Martelli, L. Sterling. Logic-Based Specification Languages for Intelligent Software Agents. Theory and Practice of Logic Programming Journal (TPLP), 4(4), Cambridge University Press, pagg. 429 – 494, 2004 `http://www.disi.unige.it/person/MascardiV/Papers/VivianaPublications.html`]

Another source of information on agents and computational logic is [M. Fisher, R. H. Bordini, B. Hirsch, P. Torroni. Computational Logics And Agents: A Road Map Of Current Technologies And Future Trends. Computational Intelligence, 23(1). 2007].

...Questions?