Agents in tuProlog

Multiagent Systems LS
Sistemi Multiagente LS

Giulio Piancastelli
`giulio.piancastelli@unibo.it`

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2007/2008

Prolog

- It is a declarative, logic programming language, not an agent programming language
  - Built-in control mechanism, not a theory of agency
- Simulates intelligence
- Simulates a degree of freedom of choice
- Performs goal-oriented operations
- Could be used to build artifacts with cognitive capabilities

Why tuProlog?

- Makes two different, complementary technologies available to build MAS abstractions
  - Java, to implement deterministic, object-oriented parts of an abstraction
  - Prolog, to create non-deterministic, logic-based parts of an abstraction
- Prolog as a language vs Java as a platform

Is this an agent?

```
start :- write('hello'), nl.
```

What is an agent?

- Agents are computational entities
- Agents are autonomous, in that
  - they encapsulate control
  - they encapsulate a criterion to govern control
- Technological vs. philosophical definition
  - Autonomy only makes sense when an individual is immersed in a society
  - No individual alone could be properly said to be autonomous

What does an agent do?

- Agents act
- An agent's action model is strictly coupled with the environment where the agent lives and acts
- Agents change the world
  - An agent can act to change the state of another agent by means of communication actions
  - An agent can act to change the state of its environment by means of pragmatical actions

A thermostat agent

```
temp(25).

start :- check_temperature

check_temperature :- temp(X), X >= 22, !, T is X - 1,
                     change_temperature(T).
check_temperature :- temp(X), X =< 18, !, T is X + 1,
                     change_temperature(T).
check_temperature :- temp(X), X > 18, X < 22.

change_temperature(X) :-
       retract(temp(_)), assert(temp(X)), !,
       check_temperature.
```

Reactivity

- Reactivity is the main new characteristic of the thermostat agent
- By means of perception of an adequate representation of the environment in which the agent is immersed, the agent can check action preconditions, or verify the effects of actions on the environment

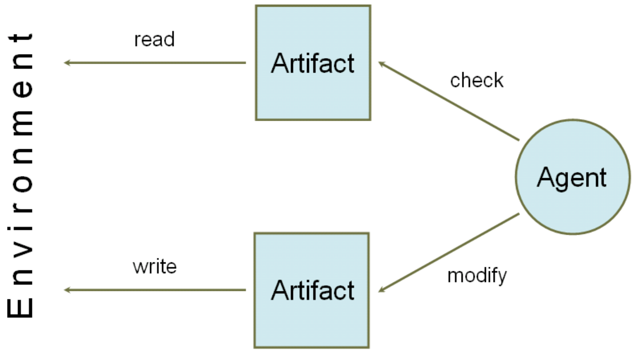
How many parts in the thermostat agent?

- An environment, conveniently represented
  - The `temp(25)` fact
- An action to directly check the state of the environment
  - The `check_temperature/0` predicate
- An action to directly modify the environment
  - The `change_temperature/1` rule

Artifacts

- Between agents and the environment, a third entity is needed to act as a mediator
- Artifacts are computational entities
- Artifacts are aimed to be used by agents
- Artifacts embody the portion of the environment that can be designed and controlled to support the system's activities
- Artifacts enable and constrain the agent's ability to manipulate the environment

System design



Environment

```
Prolog environment = new Prolog();
Theory t = new Theory("temp(25).");
environment.setTheory(t);
```

Artifacts (1/3)

```
TemperatureSensor sensor =
                new TemperatureSensor(environment);
TemperatureActuator actuator =
                new TemperatureActuator(environment);
```

Artifacts (2/3)

```
public class TemperatureSensor {
  public int getTemperature() throws Exception {
    SolveInfo query = environment.solve("temp(X).");
    if (query.isSuccess()) {
      Number temperature =
            (Number) query.getTerm("X");
      return temperature.intValue();
    } else
      return -273;
  }
}
```

Artifacts (3/3)

```
public class TemperatureActuator {
  public boolean setTemperature(int value)
                 throws Exception {
    String goal = "retract(temp(_)), " +
                  "assert(temp(" + value + ")), !.");
    SolveInfo query = environment.solve(goal);
    return query.isSuccess();
  }
}
```

Agent (1/4)

```
// create the agent
Prolog agent = new Prolog();
Theory t = new Theory(
               new FileInputStream(
                   new File("thermostat.pl")
           ));
agent.setTheory(t);
```

Agent (2/4)

```
// register artifacts within the agent
String javaLibrary =
       "alice.tuprolog.lib.JavaLibrary";
JavaLibrary library = (JavaLibrary)
                      agent.getLibrary(javaLibrary);
library.register(new Struct("sensor"), sensor);
library.register(new Struct("actuator"), actuator);
```

Agent (3/4)

```
// initialize the agent in the system
agent.solve("start.");
```

Agent (4/4)

```
start :- check_temperature

check_temperature :-
    sensor <- getTemperature returns X,
    X >= 22, !, T is X - 1, change_temperature(T).
check_temperature :-
    sensor <- getTemperature returns X,
    X =< 18, !, T is X + 1, change_temperature(T).
check_temperature :-
    sensor <- getTemperature returns X,
    X > 18, X < 22.

change_temperature(X) :-
    actuator <- setTemperature(X), check_temperature.
```

The agent model

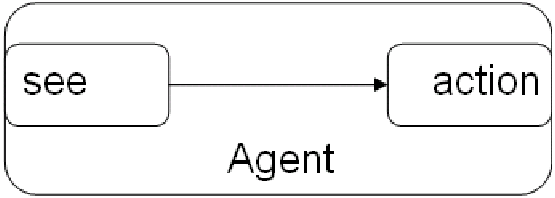


Figure: A perception agent, from the “Intelligent agents” lecture by Jeff Rosenschein

How to characterize artifacts

- In order to allow for its exploitation by agents, an artifact possibly exposes
  - a usage interface
  - operating instructions
  - a function description
- Unaware use of artifacts is likewise possible
  - The artifact's use is encoded in the agent by the programmer or designer

Mapping abstractions

- Both the environment and the agent are represented as a Prolog theory in a tuProlog engine
- Artifacts are Java objects maintaining a reference to the environment
- When entering a system, the agent is provided with references to existing artifacts, then it is started to life within that system
- Pragmatical actions are Java method calls

A cleaner agent (1/2)

```
in(1, 1).
facing(north).

dirt(2, 2).
dirt(2, 3).
dirt(3, 3).

max_x(4).
max_y(3).
```

A cleaner agent (2/2)

```
start :- go.

go :- in(X, Y), dirt(X, Y), clean(X, Y),
      facing(D), not at_borders(X, Y), !,
      move(X, Y, D), go.
go :- in(X, Y), not dirt(X, Y),
      facing(D), not at_borders(X, Y), !,
      move(X, Y, D), go.
go :- in(X, Y), at_borders(X, Y).

clean(X, Y) :- retract(dirt(X, Y)), !.

% move/3 and at_borders/2 implementation...
```

Environment vs state representations

The facts in the agent's knowledge base really describe two different parts of the system

- There is the environment representation, i.e. a description of the elements of the world where the agent lives

```
dirt(2, 2).
dirt(2, 3).
dirt(3, 3).
max_x(4).
max_y(3).
```

- There also is the *agent's internal state* representation, i.e. a description of data owned by the agent, which should be encapsulated in the agent itself and not accessible from outside

```
in(1, 1).
facing(north).
```

The agent model

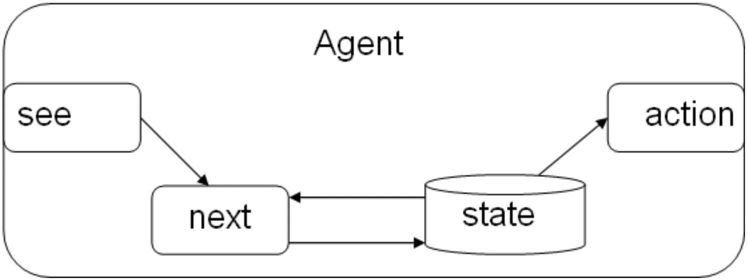


Figure: An agent with state, from the "Intelligent agents" lecture by Jeff Rosenschein

Agent

```
in(1, 1).
facing(north).

go :- in(X, Y),
      filth_sensor <- isDirty(X, Y) returns true,
      cleaner <- clean(X, Y), facing(D),
      not(border_sensor <- at_borders(X, Y) returns true),
      !, move(X, Y, D), go.
go :- in(X, Y),
      filth_sensor <- isDirty(X, Y) returns false,
      facing(D),
      not(border_sensor <- at_borders(X, Y) returns true),
      !, move(X, Y, D), go.
go :- in(X, Y),
      border_sensor <- at_borders(X, Y) returns true.
```

A multi-agent approach to the thermostat problem

To taste how multi-agent systems could be built exploting tuProlog, the simple thermostat example has been converted to using two agents

- one agent is the old thermostat agent
- the other agent is a new temperature agent, which senses the environment's temperature through the `TemperatureSensor` artifact, then communicates its value to the thermostat agent

The new challenge for the system is to let the communication between agents happen without violating any principle in the agent model and definition

Agent communication infrastructure (1/2)

- Communication between agents must preserve agent's autonomy
- Agents need to communicate through message passing rather than mechanisms like remote procedure calls
- Agents must not expose any kind of interface, not even a communication interface for other agents to talk to
- Therefore, the simple `Prolog` abstraction is no more sufficient to build a communicating agent immersed in a multi-agent system, but it needs to be encapsulated by (or coupled with) another entity, which must be able to manage the communication interface

Agent communication infrastructure (2/2)

- Communication issues are managed entirely on the Java side
- Java sockets have been used for point-to-point communication
- Since communication in the example is unidirectional (from the temperature agent to the thermostat agent) a `SocketServer` has been used on the listener side, and a simple `Socket` on the other side.
- A Java `Thread` is spawned each time a new communication connection is opened towards the listening thermostat agent, to manage communication and consequent agent's actions
- The connection port is predefined and hardcoded
- The message format is predefined and hardcoded, as well: if the message is not encoded correctly, it gets rejected

Direct vs. mediated communication

- Sockets are the raw wires of the communication infrastructure, i.e. no plugs, no gateways, no switchboards
- Communication can also happen through a *mediating artifact*, where messages are stored and agents are suspendend upon, waiting for a message of their interest
  - a mailbox for each agent
  - one or more blackboards for the whole system

The limits of the pure tuProlog approach

All the analyzed tuProlog agent systems share similar problems

- They are closed system, meaning that no new agent apart from the ones originally envisioned by the designer can enter the system
- The expressive power of abstractions available in the tuProlog system is not enough to capture the element of MAS models.
  - `Prolog` engines alone do not lead to the creation of robust MAS, not even single agents
  - `Prolog` engines are the most high-level abstraction in the system anyway
- Basic communication and coordination infrastructures need to be implemented from scratch
- Realizing other infrastructures (e.g. linkability, cognitive artifacts) could require as great an effort as for other systems

The need for broader systems

- To leverage multi-agent systems and help designers and developers, other kinds of programmable supports are needed
- tuProlog engines can be the basic bricks for those kinds of fundamental layers
  - coordination infrastructures based on a declarative, logic-based programming model
  - new logic languages providing more powerful abstractions as first class entities
  - pattern-based matching for communication facilities

Conceptual integrity

The term *conceptual integrity* has been defined by Frederick P. Brooks, Jr. in his book *The Mythical Man-Month*, published in 1975:

> *[C]onceptual integrity is* the *most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.*

Brooks also dives into the relationship between design and conceptual integrity:

> *Every part [of a system] must reflect the same philosophies and the same balancing of desiderata. Every part must even use the same techniques in syntax and analogous notions in semantics. Ease of use, then, dictates unity of desing and conceptual integrity; conceptual integrity, in turn, dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds.*

Conceptual integrity in MAS?

- To achieve conceptual integrity, a system must (always) be under total control by one or a small group of (the same) designers
- Has the plug-in architecture of Mozilla Firefox achieved conceptual integrity? Has the Web achieved conceptual integrity? Will MAS?
- As any other system, MAS absolutely need to achieve conceptual integrity at the (meta-)model level...
- ...also because nowadays it's nearly impossible to achieve conceptual integrity at the technology level.
- Just consider how many technologies are needed for the Web: server-side technologies like JSP, PHP, ASP.NET, Ruby on Rails or Django; HTML/XHTML/XML; JavaScript...
- And consider how many technologies will be needed in MAS for: agent and artifact construction and programming; environment representation; description of artifact's operations; communication between agents; message formats; discovery and immersion of agents in new systems...
- Typically, different problems are best solved by different technologies

Thanks to. . .

For having written "The Prolog Agents book" as their final project for the SID-LS course during the academic year 2005/2006, giving me some reusable material for this lecture, thanks to:

- Paolo Angelini
- Luca Boschetti
- Simone Gori
- Mattia Tonetti
- Thomas Galletti

For being an overly patient counterpart in conversations about agents, multi-agent systems and the World Wide Web model, thanks to:

- Andrea Omicini