Agent-Oriented Programming

Marco Alberti

Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

Multi-Agent Systems, 2010/2011
Outline

1. Agent-oriented programming
   - Agent0
   - PLACA

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Outline

1. Agent-oriented programming
   - Agent0
   - PLACA

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Agent-oriented programming

- Proposed by Shoham [Shoham, 1993] as “a new programming paradigm, based on a societal view of computation”.
- Main idea: directly programming agents in terms of mentalistic notions (such as beliefs, desires, intentions).
- Motivation: useful and appropriate abstraction.
Outline

1. Agent-oriented programming
   - Agent0
   - PLACa

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Agent model

Definition

An Agent0 agent is composed of:

- a set of capabilities: things the agent can do
- a set of initial beliefs
- a set of initial commitments
- a set of commitment rules
An agent can perform actions of two types:

- **private**: internally executed subroutines;
- **communicative**: messages sent to other agents.
Messages

- Messages can be of three types:
  - *request*: ask another agent to do something;
  - *unrequest*: ask another agent *not* to do something;
  - *inform*: tell an agent about something.

- Requests and unrequests usually change the addressee’s commitments; informs change the addressee’s beliefs.
Commitment rule set

- Each rule has
  - message condition
  - mental condition
  - action

- To determine if a rule fires:
  - message condition is matched against received messages
  - mental condition is matched against agent’s beliefs

- If rule fires, the agents becomes committed to the action
Agent0 commitment rule

COMMIT(
  ( agent, REQUEST, DO(time, action) ), ;;; msg condition
  ( B,
    [now, Friend agent] AND
    CAN(self, action) AND
    NOT [time, CMT(self, anyaction)]
  ), ;;; mental condition
  self,
  DO(time, action) )

Marco Alberti
Example

Paraphrase

If I receive a message from agent which requests me to do action at time, and I believe that

- agent is currently a friend,
- I can do action;
- at time, I am not committed to doing any other action,

then commit to doing action at time.
Agent0 control loop

1. Read all current messages, updating beliefs – and hence commitments – where necessary.
2. Execute all commitments for the current cycle where the capability condition of the associated action is satisfied.
Figure 1: The flow of control in Agent0.
Outline

1. Agent-oriented programming
   - Agent0
   - PLACA

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Agent0 and PLACA

- A prototype, designed to illustrate some principles, rather than be a production language.
- Serious drawback: inability of agents to plan, and communicate requests for action via high-level goals, rather than atomic actions.
- Issue addressed by PLAnning Communicating Agents (PLACA) [Thomas, 1994].
- Agents programmed in terms of mental change rules.
Example

Commitment rule

(((self ?agent REQUEST (?t (xeroxed ?x))))
  (AND (CAN-ACHIEVE (?t (xeroxed ?x))))
  (NOT (BEL (*now* shelving)))
  (NOT (BEL (*now* (vip ?agent)))))
((ADOPT (INTEND (5pm (xeroxed ?x))))
  ((?agent self INFORM
    (*now* (INTEND (5pm (xeroxed ?x)))))))
))))
Example

Paraphrase

If

- someone asks you to xerox something, and
- you can, and
- you don’t believe that they’re a VIP, or that you’re supposed to be shelving books,

then

- adopt the intention to xerox it by 5pm, and
- inform them of your newly adopted intention.
Outline

1. Agent-oriented programming
   - Agent0
   - PLACA

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Outline

1. Agent-oriented programming
   - Agent0
   - PLACA

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Concurrent MetateM

- An agent programming language based on direct execution of logical formulæ.
  - Close to the idea of agent as theorem prover.
- Several agents execute concurrently.
- Communication by asynchronous broadcast message passing.
Executable specifications

- Agent behavior specified by means of *temporal logic* formulae.
- The agent behavior is generated by *direct execution* of its specification.
- Execution corresponds to the iterative building of a *model* of the specification.
- The agent execution procedure is *correct*: if there exists a model of the specification, the agent will build it.
Iterative model building

- Common idea in computational logic: build a model for a set of formulæ by
  - starting from an empty *interpretation*
  - at each step, extending the interpretation to satisfy the formulæ that are false.
- If formulæ are implications, this process can be seen as “firing” of rules.
Consider the set of formulæ

\[
\begin{align*}
 & a \\
 & a \rightarrow b \\
 & c \rightarrow d \\
 & a \land b \rightarrow e
\end{align*}
\]

(1)

Iterative model building proceeds as follows:

1. \{\}\n2. \{a\}\n3. \{a, b\}\n4. \{a, b, e\}
Agent architecture

Definition

A MetateM agent is composed of:

- an *agent interface*, which defines how the agent interacts with its environment (i.e., other agents); and
- a *computational engine*, which defines how the agent acts

The computational engine is Based on the MetateM paradigm of *executable temporal logic* [Barringer et al., 1989].
Agent interface

**Definition**

An *agent interface* is composed of

- an *agent identifier*: unique name for the agent;
- *environment propositions*: a set of symbols defining which messages will be accepted by the agent; and
- *component propositions*: a set of symbols defining the messages that the agent may send.

**Agent interface**

```
stack(pop, push)[popped, full]
```
Agent specification

- Uses Propositional MetateM Logic (PML): propositional logic augmented with temporal operators (see next slide).
- The specification is given by means of PML formulæ.

**Definition**

A **program rule** is a PML formula of the form

antecedent about past $\implies$ consequent about present and future.

- Whenever the agent detects that the antecedent is true, it tries to make the consequent true too (iterative model building).
- “Declarative past, imperative future” [Gabbay, 1987].
### Temporal connectives

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\bigcirc \varphi$</td>
<td>$\varphi$ is true tomorrow</td>
</tr>
<tr>
<td>$\bigcirc \varphi$</td>
<td>$\varphi$ was true yesterday</td>
</tr>
<tr>
<td>$\lozenge \varphi$</td>
<td>at some time in the future, $\varphi$</td>
</tr>
<tr>
<td>$\Box \varphi$</td>
<td>always in the future, $\varphi$</td>
</tr>
<tr>
<td>$\lozenge \varphi$</td>
<td>at some time in the past, $\varphi$</td>
</tr>
<tr>
<td>$\blacksquare \varphi$</td>
<td>always in the past, $\varphi$</td>
</tr>
<tr>
<td>$\varphi U \psi$</td>
<td>$\varphi$ will be true until $\psi$</td>
</tr>
<tr>
<td>$\varphi S \psi$</td>
<td>$\varphi$ has been true since $\psi$</td>
</tr>
<tr>
<td>$\varphi W \psi$</td>
<td>$\varphi$ is true unless $\psi$</td>
</tr>
<tr>
<td>$\varphi Z \psi$</td>
<td>$\varphi$ is true zince $\varphi$</td>
</tr>
</tbody>
</table>

*Table 1: Propositional MetateM Logic operators*
Temporal connective meanings

In the present state,

- $\Diamond \varphi$ (resp. $\Diamond \varphi$) is true iff $\varphi$ is true in the next (resp. previous) state.

- $\varphi$ (resp. $\varphi$) is true iff $\varphi$ is true in some future (resp. past) state. *including* the present state.

- $\Box \varphi$ (resp. $\Box \varphi$) is true iff $\varphi$ is true in all future (resp. past) states, *including* the present state.

- $\varphi U \psi$ (resp. $\varphi S \psi$) is true iff $\psi$ is true in some future (resp. past) state, and $\varphi$ is true until (resp. since) then.

- $\varphi W \psi$ (resp. $\varphi Z \psi$) is true iff $\varphi$ is true in all future (resp. past) states until (resp. since) $\psi$ is true, which may never happen.
Temporal connective examples

**Always**

□ *important*(agents)

“Agents are important now, and they will always be”.

**Sometime in the future**

◊ *important*(Janine)

“Sometime in the future, Janine will be important”.

Marco Alberti
Temporal connective examples

**Until**

\[ \neg \text{friends}(\text{us}) \cup \text{apologize}(\text{you}) \]

“We are not friends until you apologize”.

**Tomorrow**

\[ \circ \text{apologize}(\text{you}) \]

“Tomorrow (in the next state) you shall apologize”.
Update the agent’s *history* by receiving messages (those that match with the agent’s environment propositions)

2. Check which rules *fire*, comparing each rule’s antecedent with the current history

3. *Jointly execute* the fired rules with *commitments* from previous cycles (see next slide)

4. Goto 1
Execution of fired rules and commitments

- Collect the consequents of newly fired rules with old commitments: these become the new constraints.
- Try to create the next state satisfying such constraints.
- Note that:
  - it may not be possible to satisfy all constraints creating the next states; in this case, unsatisfied commitments are carried over to the next state;
  - current constraints are a disjunctive formula: the agent must choose among a number of execution possibilities.

Disjunctive constraints

- $\Diamond a$ is satisfied by making $a$ true in any of the future states;
- $\neg \Box a$ is satisfied by making $a$ false in any of the future states.
Communication

- When a proposition becomes true, it is checked against the agent’s component propositions: if it matches, it is broadcast as a message.
- On receipt of a message, each agent check it against its environment propositions: if it matches, the agent adds it to its history.
Sample Concurrent MetateM specification

- Simple resource allocation scenario.
- Three agents:
  - *rp*, resource producer;
  - *rc1* and *rc2*, resource consumers.
Sample Concurrent MetateM specification

Producer

- Receives \textit{ask1}, \textit{ask2}.
- Sends \textit{give1}, \textit{give2}.
- Can only give a resource to an agent at a time, and will eventually give the resource to an agent that asks for it.

\textit{rp} specification

\[\textit{rp}(\textit{ask1}, \textit{ask2})[\textit{give1}, \textit{give2}] :\]

- \textit{ask1} \rightarrow \Diamond \textit{give1}
- \textit{ask2} \rightarrow \Diamond \textit{give2}
- \textit{start} \rightarrow \Box \neg (\textit{give1} \land \textit{give2})\]
Sample Concurrent MetateM specification

First consumer

- Receives `give1`.
- Sends `ask1`.
- Asks for the resource at every cycle.

\[ rc1 \text{(give1)}[\text{ask1}]: \]
- `start` \(\rightarrow\) `ask1`
- `\text{ask1}` \(\rightarrow\) `ask1`
Sample Concurrent MetateM specification

Second consumer

- Receives *ask1*, *give2*.
- Sends *ask2*.
- Asks for the resource at each cycle following one where *rc1* asked for the resource and *rc2* did not.

**rc2 specification**

\[ rc2(ask1, give2)[ask2] : \quad \diamond(ask1 \land \neg ask2) \rightarrow ask2 \]
### Sample run

<table>
<thead>
<tr>
<th>Time</th>
<th>Agent</th>
<th>Agent</th>
<th>Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>rp</strong></td>
<td><strong>rc1</strong></td>
<td><strong>rc2</strong></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td><em>ask1</em></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td><em>ask1</em></td>
<td><em>ask1</em></td>
<td><em>ask2</em></td>
</tr>
<tr>
<td>2</td>
<td><em>ask1, ask2, give1</em></td>
<td><em>ask1</em></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><em>ask1, give2</em></td>
<td><em>ask1, give1</em></td>
<td><em>ask2</em></td>
</tr>
<tr>
<td>4</td>
<td><em>ask1, ask2, give1</em></td>
<td><em>ask1</em></td>
<td><em>give2</em></td>
</tr>
<tr>
<td>5</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Outline

1. Agent-oriented programming
   - Agent0
   - PLACA

2. Concurrent MetateM
   - The Concurrent MetateM agent programming language
   - Snow White in Concurrent MetateM
Snow White example

- Toy example
- Exemplifies agent characterization by means of logical relations over intentional notions.
- Agents:
  - Snow White
  - The modified seven dwarfs: Eager, Mimic, Jealous, Greedy, Courteous, Generous, Shy
- Messages:
  - \textit{ask}(x): dwarf \( x \) asks Snow White for a sweet;
  - \textit{give}(x): Snow White gives dwarf \( x \) a sweet.

(note variables)
Eager

Eager agent

- Eager initially asks for a sweet and, from then on, whenever he receives a sweet, asks for another.

\[
eager(give)[ask] : \\
\text{start} \rightarrow ask(eager) \\
\circ give(eager) \rightarrow ask(eager)
\]
Mimic

Mimic agent

- Mimic asks for a sweet whenever he sees eager asking for one.

\[
mimic(\text{ask})[\text{ask}] : \langle \text{ask(eager)} \rightarrow \text{ask(mimic)} \rangle
\]
Jealous

Jealous asks for a sweet whenever he sees eager receiving one.

\[
\text{jealous}(\text{give})[\text{ask}] : \\
\diamond \text{give}(\text{eager}) \rightarrow \text{ask}(\text{jealous})
\]
Greedy

Greedy agent

Greedy asks for a sweet as often as he can.

\[
greedy()[\text{ask}]: \\
start \rightarrow \Box \text{ask}(\text{greedy})
\]
Courteous

Courteous asks for a sweet only when eager, mimic, jealous and greedy have all asked for one, since courteous last asked.

\[
\text{courteous}(\text{ask})[\text{ask}] : \\
(\neg \text{ask}(\text{courteous})) \text{Sask}(\text{eager}) \land \\
(\neg \text{ask}(\text{courteous})) \text{Sask}(\text{mimic}) \land \\
(\neg \text{ask}(\text{courteous})) \text{Sask}(\text{jealous}) \land \\
(\neg \text{ask}(\text{courteous})) \text{Sask}(\text{greedy})) \rightarrow \text{ask}(\text{courteous})
\]
Generous

**Generous agent**

- Generous asks for a sweet only when eager, mimic, jealous, insistent and courteous have all received one, since generous last asked for one.

\[
generous(give)[ask] : \\
((\neg ask(generous))S_{give}(eager) \land \\
(\neg ask(generous))S_{give}(mimic) \land \\
(\neg ask(generous))S_{give}(jealous) \land \\
(\neg ask(generous))S_{give}(greedy) \land \\
(\neg ask(generous))S_{give}(courteous)) \rightarrow ask(generous)
\]
Shy

Shy agent

- Shy only asks for a sweet when he sees no one else asking.

\[
\begin{align*}
\text{shy(ask)[ask]} & : \\
\text{start} & \rightarrow \Diamond \text{ask(shy)} \\
\text{ask(x)} & \rightarrow \neg \text{ask(shy)} \\
\text{ask(shy)} & \rightarrow \Diamond \text{ask(shy)}
\end{align*}
\]
Snow White

SnowWhite agent

- Snow White has some sweets (resources), which she will give to the dwarfs (resource consumers).
- She will only give to one dwarf at a time.
- She will always eventually give to a dwarf that asks.

\[
\text{snowwhite}(\text{ask})[\text{give}] : \\
\quad \Diamond \text{ask}(x) \rightarrow \diamond \text{give}(x) \\
\quad \text{give}(x) \land \text{give}(y) \rightarrow x = y
\]
Metatem: A framework for programming in temporal logic.
In de Bakker, J. W., de Roever, W. P., and Rozenberg, G., editors, 

The declarative past and imperative future: Executable temporal logic for interactive systems.