High Level, Multi-Agent Prototypes from a Scenario-Path Notation: A Feature-Interaction Example

http://www.sce.carleton.ca/rads/agents/

R.J.A. Buhr, D. Amyot, M. Elammari, D. Quesnel
Carleton University, Ottawa, Canada
buhr@sce.carleton.ca, damyot@csi.uottawa.ca, {elammari, quesnel}@scs.carleton.ca

T. Gray, S. Mankovski
Mitel Corporation, Kanata, Canada
{tom_gray, serge_mankovski}@mitel.com
The Essence

- Use Case Maps (UCMs) provide understandable visual descriptions of *agencies*, in the form of path diagrams that combine structure and behavior. Normally-difficult-to-understand agency-wide dynamic situations are explicitly represented, exposing possible intra-agency conflicts.

- BDI-style agent metamodels (represented by tables) flow in a systematic way from UCMs.

- Executable agency prototypes (implemented with concurrent CLIPS rule engines, synchronizing via blackboards) flow in a systematic way from agent tables, enabling study of agency behaviour without becoming overwhelmed by details.

- Telephony with agents provides a practical example (features are added dynamically and feature interactions must be resolved dynamically).
Road Map

To begin with concepts familiar to this audience, the approach is presented bottom up:

- Introduction to the running example of telephony feature interaction.
- Executable prototypes of telephony agencies, demonstrating feature interaction and its resolution.
- Tables for agent metamodels; prototypes from tables
- UCMs for agency description; tables from UCMs.
- Summary of the approach, from the top down.
- Scalability, practicality, status.
- Conclusions.
Example of feature interaction in telephony

I want to call B...

A calls B

B forwards to X

OCS: calls to X are forbidden

CF: forward all calls to X

OCS = originating call screening

CF = call forwarding

An interesting variation:

A calls A

A forwards to X
Agency prototypes

Blackboard provides a universal coordination environment:

- between agents
- between rule engines in agent heads
- between agent heads and agent bodies
- between agents and Mediapath
Competing Rule Engines Communicating via a Blackboard

Call-Side (A)

{ :connectFrom A :connectTo B :callID id }
{ :callFrom A :callTo B :callID id :status PERMIT }
{ :callFrom A :callTo B :callID id :status PERMIT }
{ :callFrom A :callTo B :callID id :status Tuple(:FORWARDTO X) }

ORIGINATING

OCS

{ :connectFrom A :connectTo X :callID id2 }
{ :callFrom A :callTo X :callID id2 :status PROHIBIT }
.... <responses from X are not important>
{ :cancelCall id2 }

Answer-Side (B)

Micmac Blackboard

TERMINATING

CF

Slide 6
CLIPS Rules for OCS and CF

a) Originating Call Screening

(defrule prohibit
  (declare (salience 2))
  ?c <- (callTo ?to)
  (prohibit ?to)
=>
  (doProhibit)
  (retract ?c))

(defrule permit
  (declare (salience 1))
  ?c <- (callFrom ?from)
=>
  (doPermit)
  (retract ?c))

b) Call Forwarding

(defrule forward
  ?c <- (call)
  (forward ?to)
=>
  (doForward ?to)
  (retract ?c))

• Rules are ordered according to their salience before execution.
(Shows a call prohibited by OCS directly)
Micmac & Mediapath Prototype Screen (2)

(Shows a call prohibited by OCS through CF)
### Agent Internal Model

Describes agents in terms of their goals, beliefs and tasks.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Precondition</th>
<th>Postcondition</th>
<th>Task</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process originating call</td>
<td>Number is collected</td>
<td>Request sent to answerer</td>
<td>send_request</td>
<td>ORIGINATING</td>
</tr>
<tr>
<td>Process originating call</td>
<td>Outgoing call connection requested</td>
<td>Call permitted or rejected</td>
<td>check_list doPermit doReject</td>
<td>OCS</td>
</tr>
<tr>
<td>Process call request</td>
<td>There is an incoming call</td>
<td>Caller and/or answerer are notified</td>
<td>ring notify_caller</td>
<td>TERMINATING</td>
</tr>
<tr>
<td>Process call request</td>
<td>CF is on. There is an incoming call</td>
<td>Caller notified of a new destination</td>
<td>doForward</td>
<td>CF</td>
</tr>
</tbody>
</table>

- Each Plan (row) is to be implemented as a *Clips engine*.
- Goals, preconditions, and postconditions are *facts*.
- Tasks are *functions*.
**Conversational Model**

Describes the coordinations among the agents. It basically identifies what messages are exchanged in order for agents to cooperate and negotiate with each other to fulfill inter-agent dependencies.

<table>
<thead>
<tr>
<th>Received</th>
<th>Sent</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prop(:connectFrom a :connectTo b)</td>
<td>ORIGINATING</td>
</tr>
<tr>
<td>2</td>
<td>Prop(:connectFrom a :connectTo b)</td>
<td>ACCEPT</td>
</tr>
<tr>
<td>3</td>
<td>Prop(:connectFrom a :connectTo b)</td>
<td>ACCEPT</td>
</tr>
<tr>
<td>4</td>
<td>Prop(:connectFrom a :connectTo b)</td>
<td>CProp(:connectFrom a :connectTo f)</td>
</tr>
<tr>
<td>5</td>
<td>CProp(:connectFrom a :connectTo f)</td>
<td>Prop(:connectFrom a :connectTo f)</td>
</tr>
</tbody>
</table>

- The conversational model is implemented by tuples.
- Agents exchange tuples using the Micmac blackboard.
From Agent Models to Tuples and CLIPS Engines

Row 4 of table for agent internal model

<table>
<thead>
<tr>
<th>Goal</th>
<th>Precondition</th>
<th>Postcondition</th>
<th>Task</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process call request</td>
<td>CF is on.</td>
<td>Caller notified of a new destination</td>
<td>doForward</td>
<td>CF</td>
</tr>
<tr>
<td></td>
<td>There is an incoming call</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Micmac blackboard

```
{ :callFrom A :callTo B :callID id :status Tuple(:FORWARDTO X) }
```

Tuple format follows from conversational model

Row 4 of table for conversational model

<table>
<thead>
<tr>
<th>Received</th>
<th>Sent</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prop(:connectFrom a :connectTo b)</td>
<td>CProp(:connectFrom a :connectTo f)</td>
<td>CF</td>
</tr>
</tbody>
</table>

Clips rule follows from agent internal model

```
(defrule forward
  ?c <- (call)
  (forward ?to)
  =>
  (doForward ?to)
  (retract ?c))
```

Rule engine asserts tuple
A Telephony-Agency UCM

Slide 13
**UCM notation**

**NOTATION**
- **stub**
- **dynamic stub**
- **OR-join**
- **OR-fork**
- **AND-fork**
- **path start**
- **path end**
- **responsibility names along paths**
- **busy**
- **concurrent paths**

**CONVENTIONS OF THIS DIAGRAM**

STUB FORM FOR CSP AND ASP
(CSP = Caller-Side Processing
ASP = Answerer-Side Processing)

- **diversion**
- **normal entry**
- **normal exit**
- **status out**
Selected scenarios

A connects to B.

A tries and fails to connect to X.
OCS-CF feature interaction and resolution

Feature interaction: A connects to X through CF.

Redesign UCM to get rid of interaction (reflects earlier implementation).
From UCMs to agent metamodels

Inter-agent path segments imply agent coordination.

Relationships are general ones applicable to agent metamodels that may differ in detail.
Summary of the approach, from the top down

• Constructive approach.
• UCMs used for discovery and understanding.
• Tables for agent metamodels flow systematically from UCMs.
• Abstract prototypes flow systematically from tables.
• Prototypes focus on problems of control and coordination in dynamic agencies, not application functionality.
Scalability, practicality

• Experience indicates that adding features does not cause UCM models to blow up.

• Agents inherently handle feature interactions in a scalable way by divide and conquer; agents allow us to build systems, local parts of which can be customized, personalized and even evolved without reference to other parts.

• The prototyping approach of competing rule engines is flexible and scalable.

• Execution requires a coordination environment, not specifically blackboards; other approaches may be used for practical implementation, or technology may make blackboards practical.

• Extensions to the BDI model (e.g., OPI) are planned to enhance control of dynamic situations at run time.
Conclusions

- The dynamic nature of agent solutions presents great difficulties for telecom software designers.
- Our contribution consists in bridging the large conceptual gap between dynamic situations involving multiple agents and software design details of individual agents.
- UCMs are used to view and understand complicated dynamic situations.
- UCMs are translatable into BDI-style solutions, and into high level, executable prototypes.
- The prototype environment enables issues such as dynamic conflict resolution to be explored.
- The practicality of the approach is illustrated by a practical problem: telephony feature interactions.
- We offer a scalable approach that permits adding new features incrementally at run-time.