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

http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/4paam98.pdf

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*

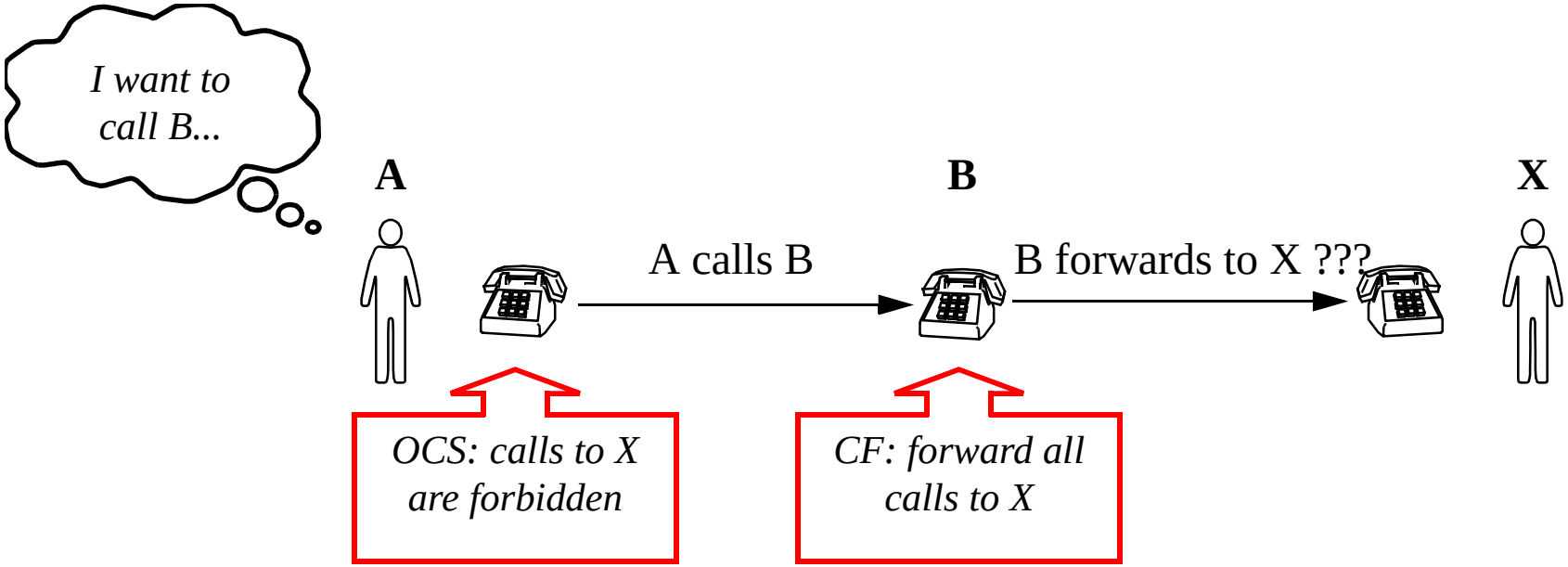
- Use Case Maps (UCMs) provide understandable visual descriptions of *agencies*, in the form of path diagrams that combine structure and behavior. Normally-difficult-to-understand agency-wide dynamic situations are explicitly represented, exposing possible intra-agency conflicts.
- BDI-style agent metamodels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

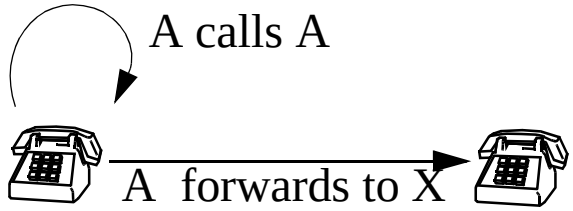
# *Example of feature interaction in telephony*



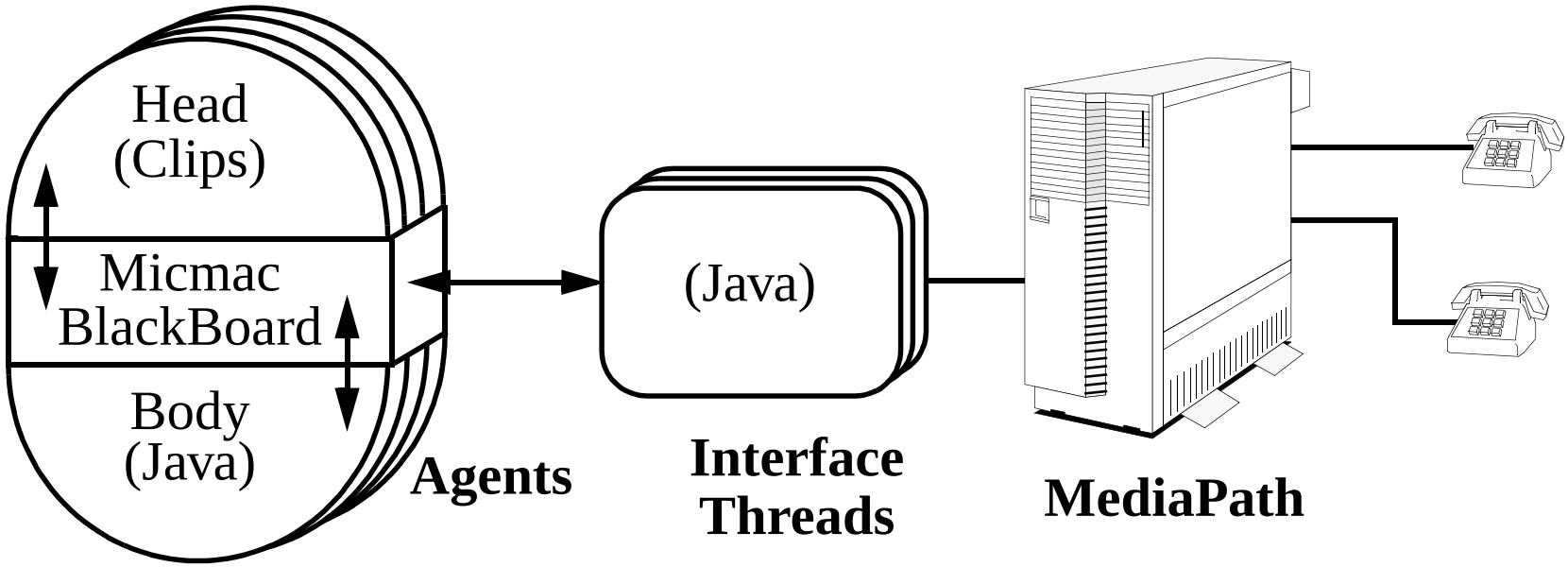

OCS = originating call screening

CF = call forwarding

An interesting variation:



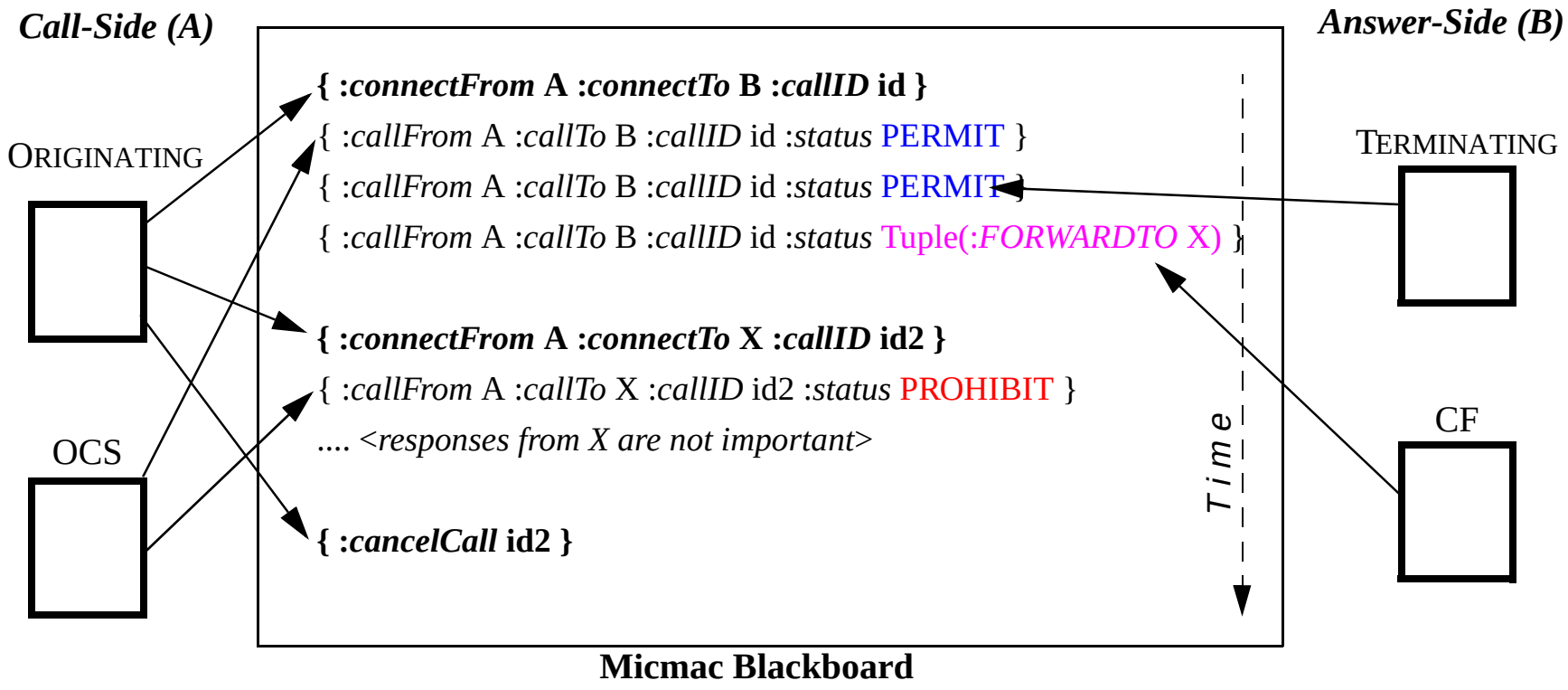
# *Agency prototypes*




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*



# *CLIPS Rules for OCS and CF*

**a) Originating Call Screening**

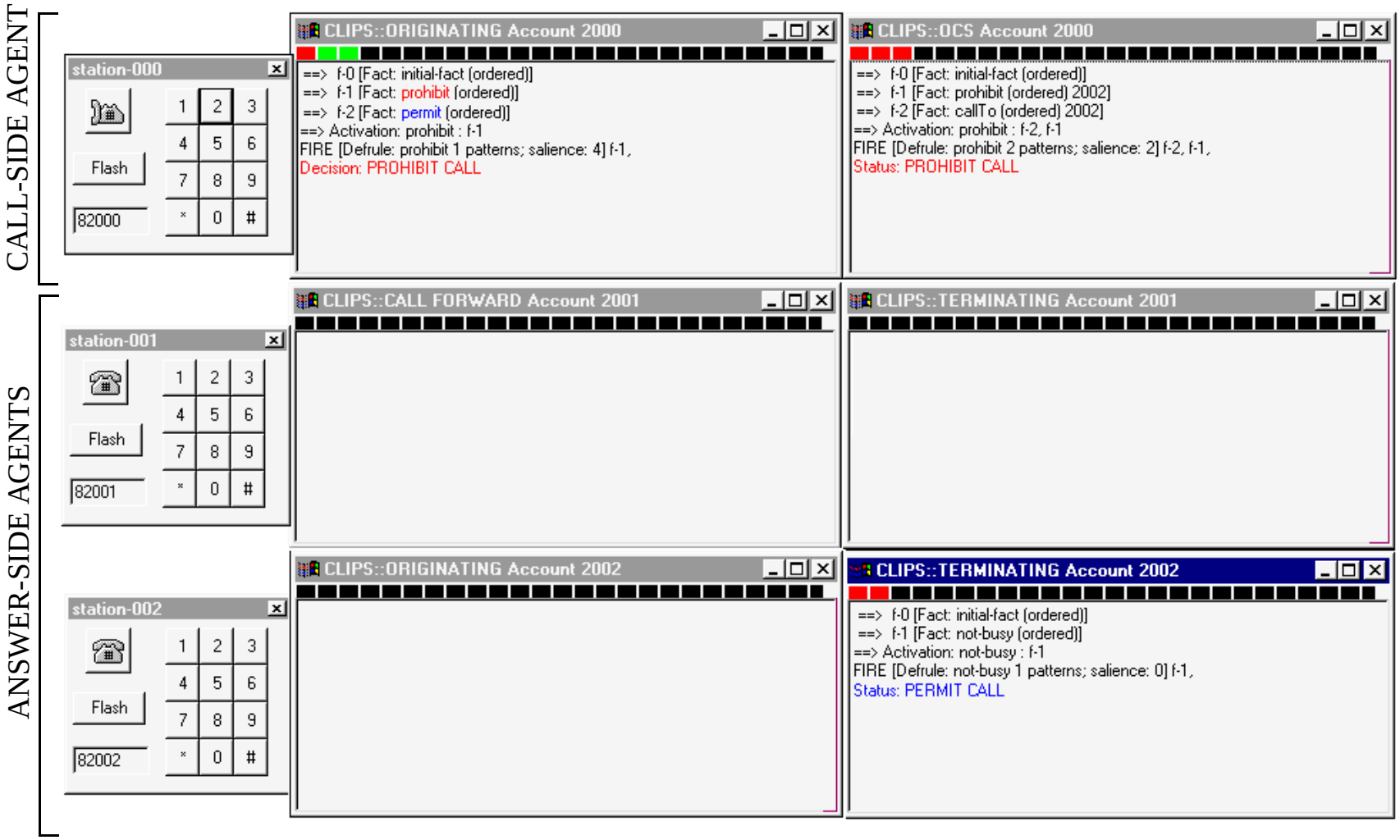
```
(defrule prohibit
      (declare (salience 2))
      ?c <- (callTo ?to)
      (prohibit ?to)
=>
      (doProhibit)
      (retract ?c))

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

**b) Call Forwarding**

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

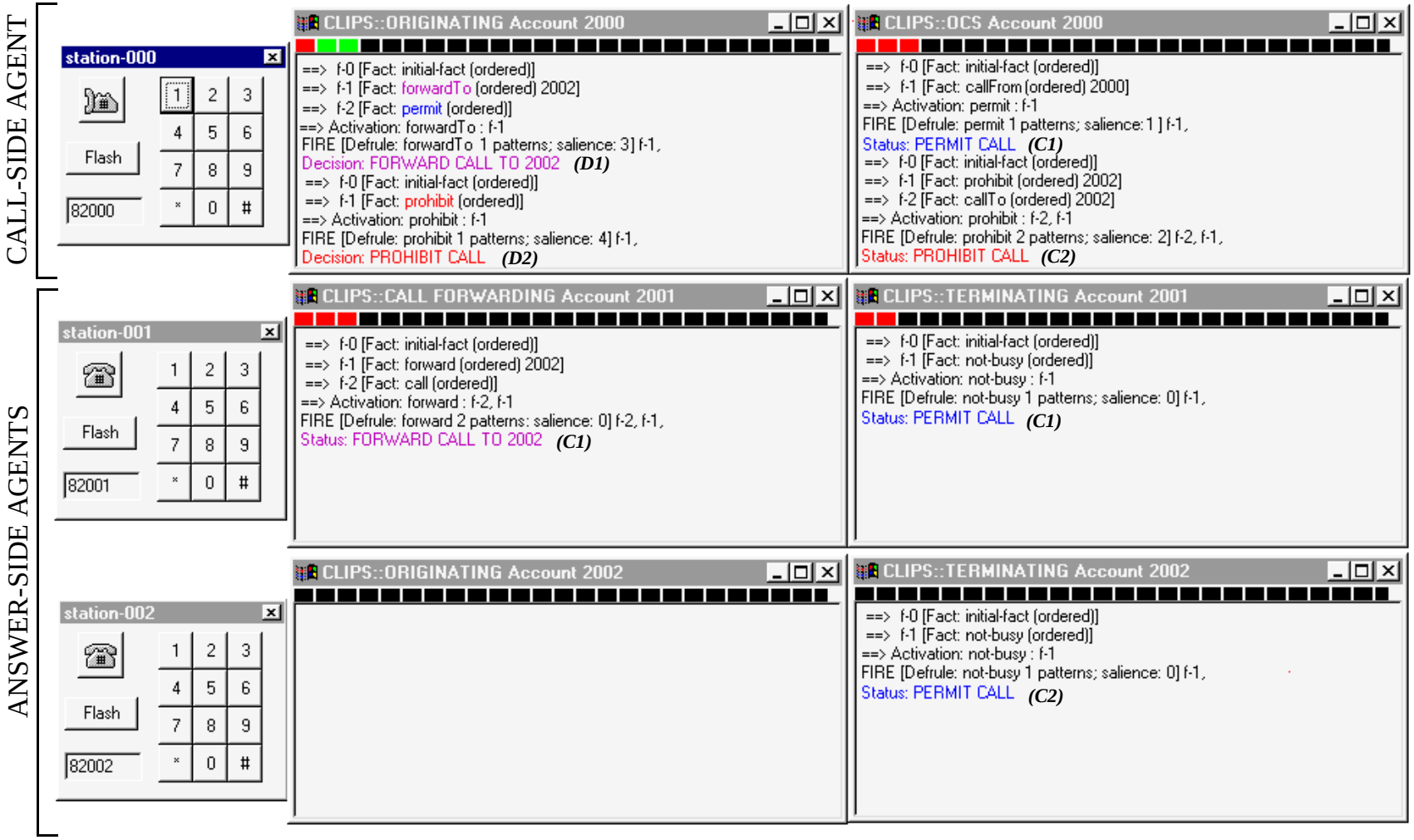
- Rules are ordered according to their ***salience*** before execution.

# *Micmac & Mediapath Prototype Screen (1)*




( 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.

| | Goal | Precondition | Postcondition | Task | Comment |
|---|---|---|---|---|---|
| 1 | Process originating call | Number is collected | Request sent to answerer | send_request | ORIGINATING |
| 2 | Process originating call | Outgoing call connection requested | Call permitted or rejected | check_list<br>doPermit<br>doReject | OCS |
| 3 | Process call request | There is an incoming call | Caller and/or answerer are notified | ring<br>notify_caller | TERMINATING |
| 4 | Process call request | CF is on.<br>There is an incoming call | Caller notified of a new destination | doForward | CF |

- 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.

| | Received | Sent | Comment |
|---|---|---|---|
| 1 | | Prop(:connectFrom a :connectTo b) | ORIGINATING |
| 2 | Prop(:connectFrom a :connectTo b) | ACCEPT \| REJECT | OCS |
| 3 | Prop(:connectFrom a :connectTo b) | ACCEPT \| REJECT | TERMINATING |
| 4 | Prop(:connectFrom a :connectTo b) | CProp(:connectFrom a :connectTo f) | CF |
| 5 | CProp(:connectFrom a :connectTo f) | Prop(:connectFrom a :connectTo f) | ORIGINATING |

- 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*

| Goal | Precondition | Postcondition | Task | Comment |
|---|---|---|---|---|
| Process call request | CF is on. There is an incoming call | Caller notified of a new destination | doForward | CF |

***Clips rule follows from agent internal model***

***Rule engine asserts tuple***

*Micmac blackboard*

...

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

**CF**

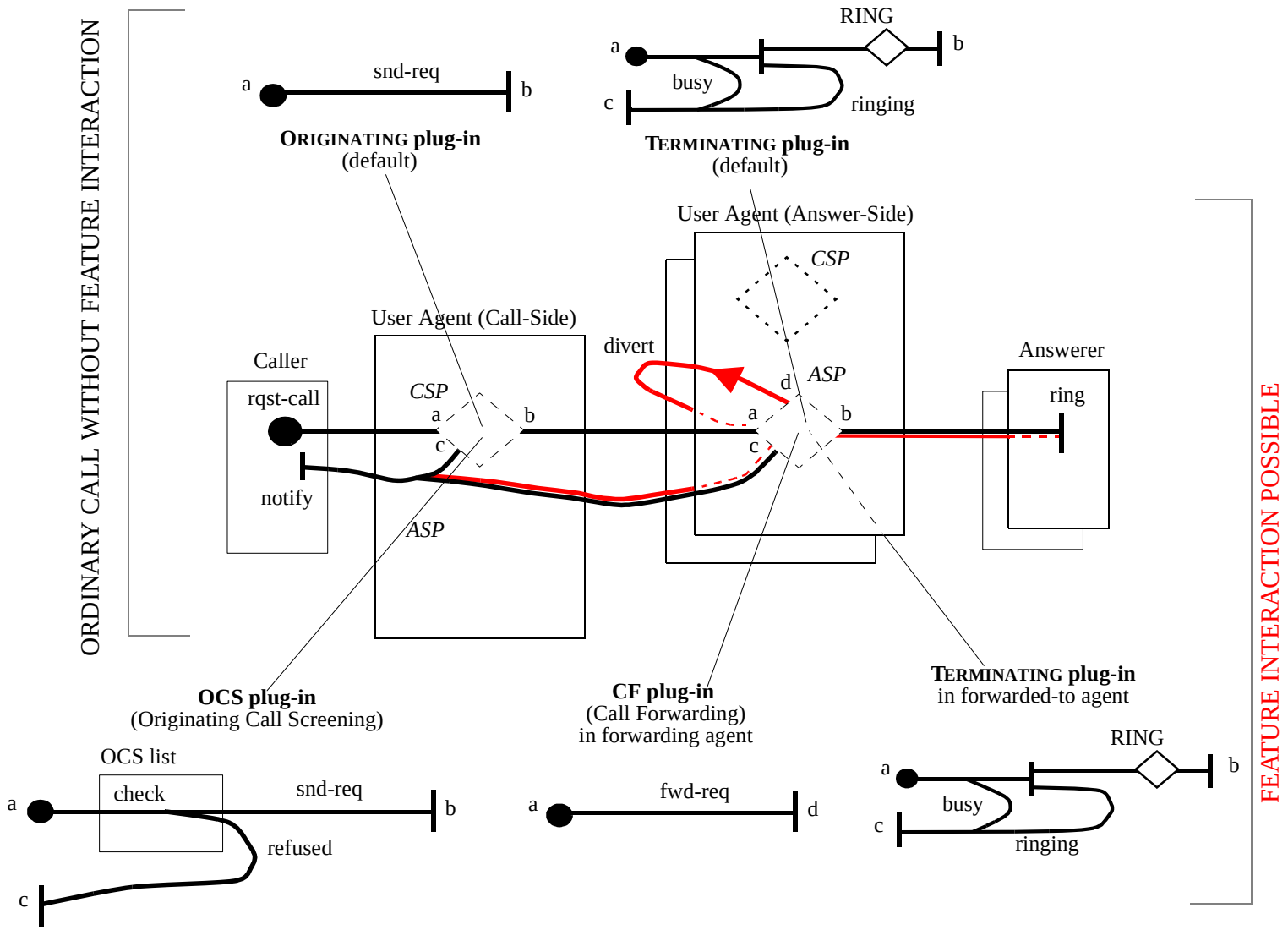
```
(defrule forward
   ?c <- (call)
   (forward ?to)
=>
   (doForward ?to)
   (retract ?c))
```

***Tuple format follows from conversational model***

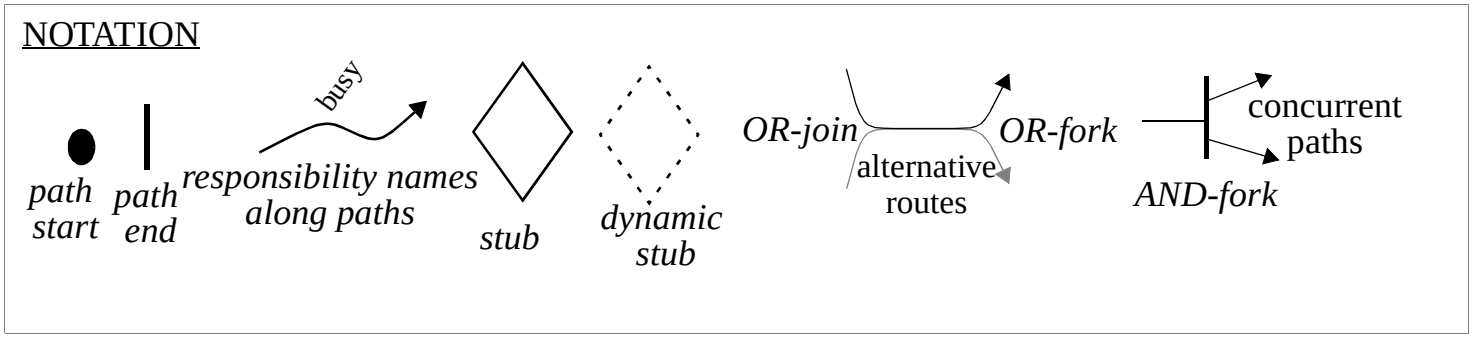
*Row 4 of table for conversational model*

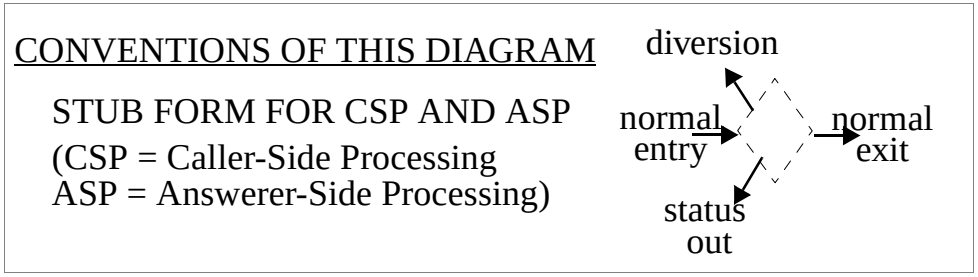
| Received | Sent | Comment |
|---|---|---|
| Prop(:connectFrom a :connectTo b) | CProp(:connectFrom a :connectTo f) | CF |

# A Telephony-Agency UCM

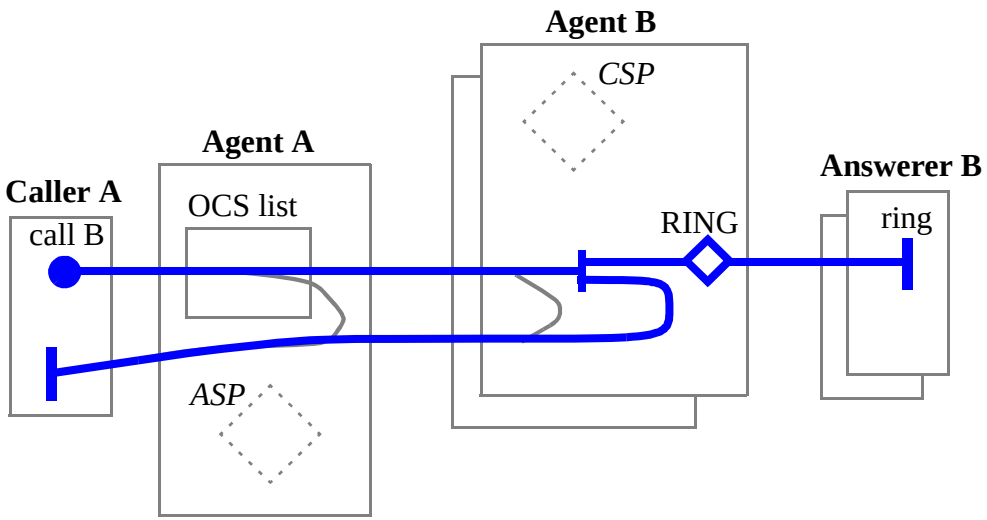


# UCM notation

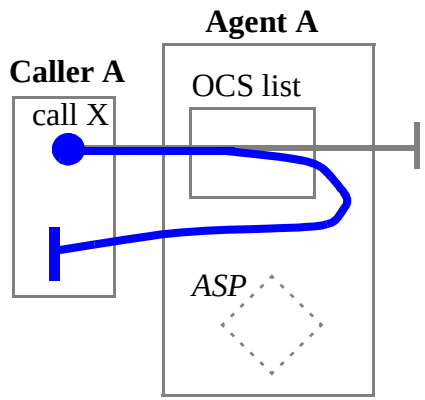


# *Selected scenarios*

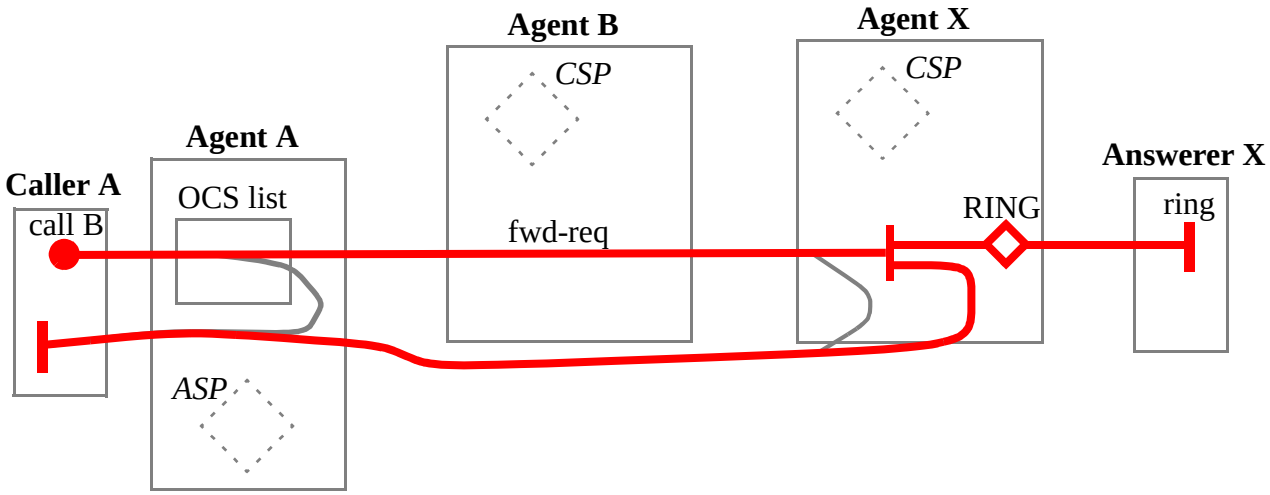



**A connects to B.**

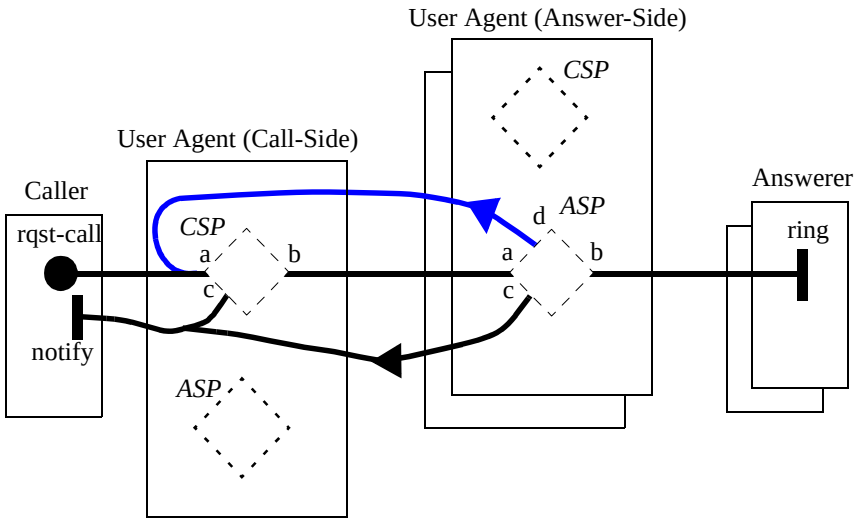



**A tries and fails to connect to X.**

# *OCS-CF feature interaction and resolution*

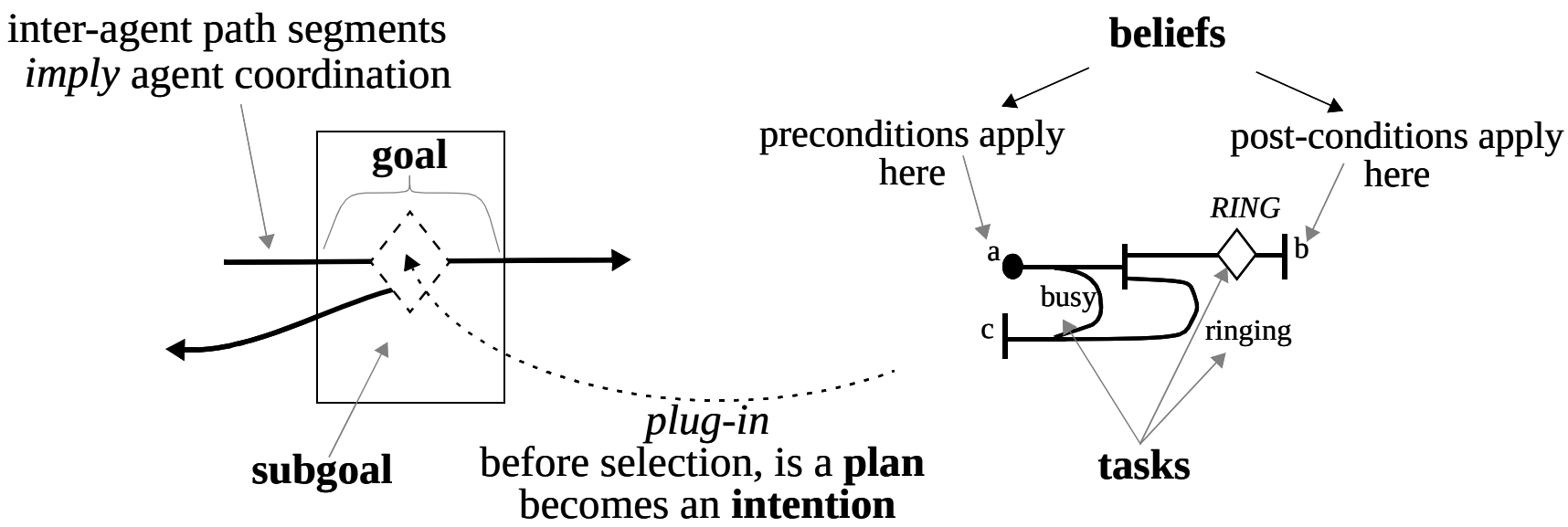



**Feature interaction: A connects to X through CF.**




**Redesign UCM to get rid of interaction (reflects earlier implementation).**

# *From UCMs to agent metamodels*




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.