Feature-Interaction Visualization and Resolution in an Agent Environment

http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/fiw98.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

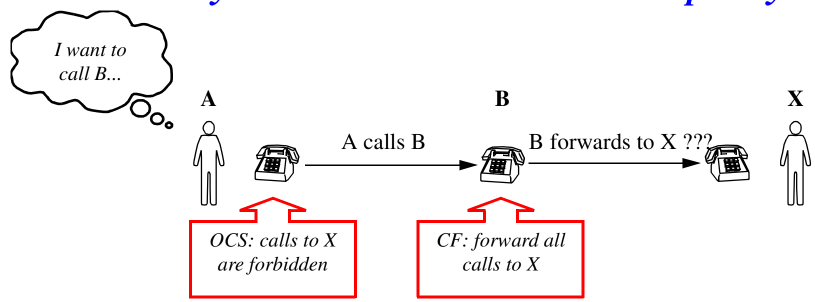
- Use Case Maps (UCMs) provide visual descriptions of *agencies*, in the form of path diagrams that combine structure and behavior. Agency-wide dynamic situations are explicitly represented, exposing possible intra-agency conflicts.
- Agent metamodels (represented by tables) flow in a systematic way 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 interactions must be resolved dynamically. PC-based systems coupled to enterprise processes with global policies, which will cause interactions that go beyond the classical ones, are coming...

Road Map

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

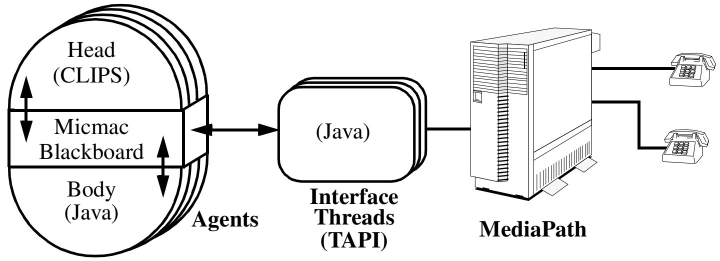
- Introduction to the running example of telephony feature interaction.
- Executable prototypes of telephony agencies, demonstrating feature interaction and its resolution.
- Tables for agent metamodels; prototypes from tables
- UCMs for agency description; tables from UCMs.
- Summary of the approach,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

Case Study: Feature Interactions in Telephony



- Telecom systems are a fertile field for agents.
- New features are added, and conflicts must be resolved, often dynamically.
- Agents offer the prospect of incrementally adapting to such changes.
- Need for a bridge between the big picture and details of individual agents

Architecture for Agency Prototypes

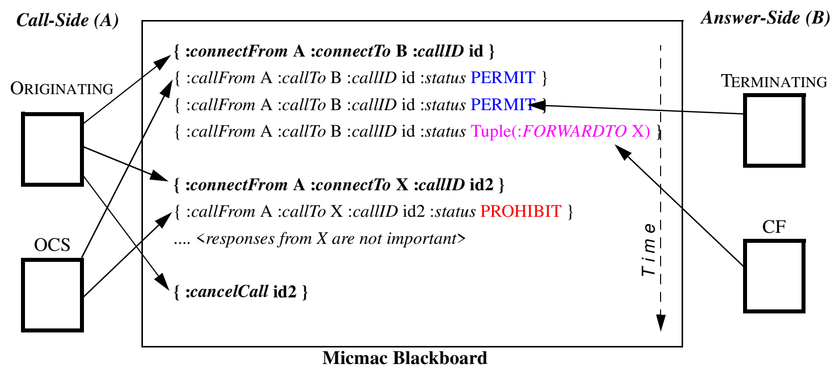


Blackboard provides a universal coordination environment:

- between agents
- between rule engines in agent heads
- between agent heads and agent bodies
- between agents and ***Mediapath*** (http://www.mitel.com/MediaPath)

Coordination and negotiation similar in spirit to the approach of [Griffeth and Velthuijsen, 1994]

Competing Rule Engines Communicating via a Blackboard



(http://micmac.mitel.com)

CLIPS Rules for OCS and CF

a) Originating Call Screening

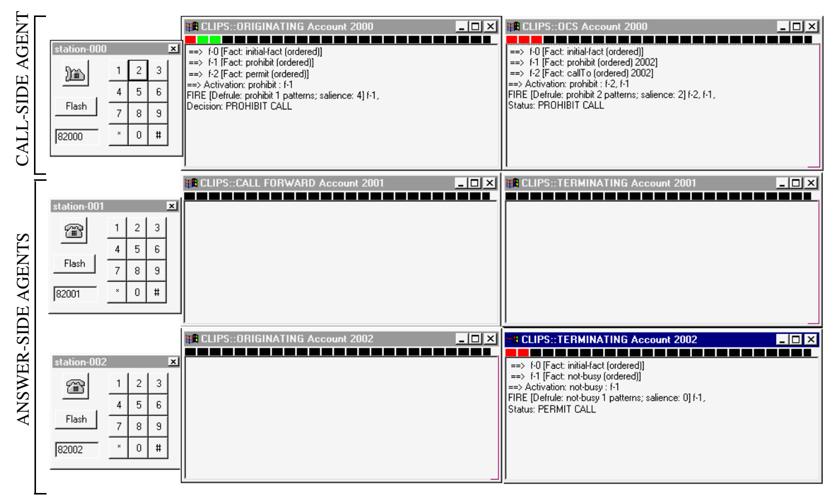
```
(defrule prohibit
      (declare (salience 2))
      ?c <- (callTo ?to)
      (prohibit ?to)
=>
      (doProhibit)
      (retract ?c))

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

b) Call Forwarding

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

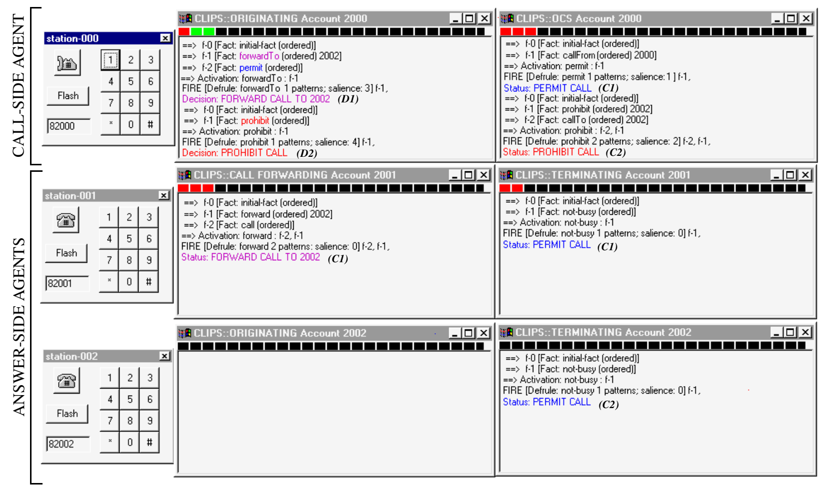
- Rules are ordered according to their ***salience*** before execution.

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 doPermit doReject	OCS
3	Process call request	There is an incoming call	Caller and/or answerer are notified	ring notify_caller	TERMINATING
4	Process call request	CF is on. 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*)	PERMIT | PROHIBIT	OCS
3	Prop(:connectFrom *a* :connectTo *b*)	PERMIT | PROHIBIT	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))
```

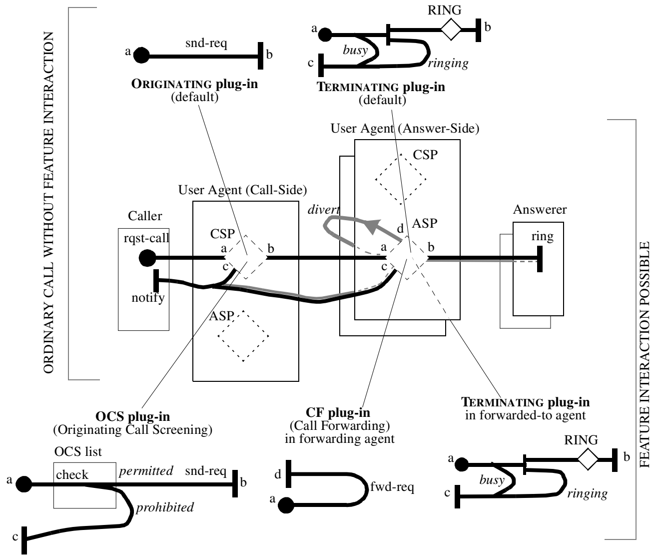
Tuple format follows from conversational model

Row 4 of table for conversational model

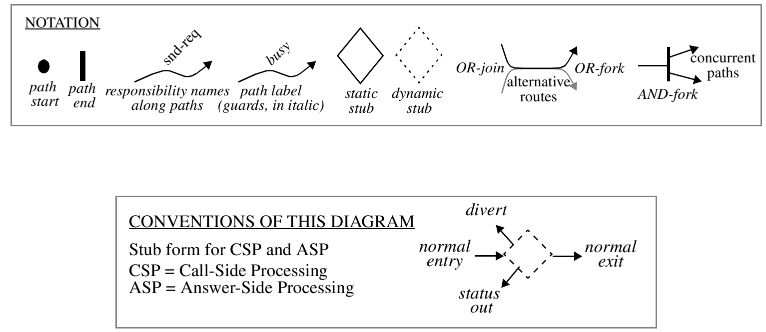
Received	Sent	Comment
Prop(:connectFrom *a* :connectTo *b*)	CProp(:connectFrom *a* :connectTo *f*)	CF

Conversational model: direct negotiation between agents.

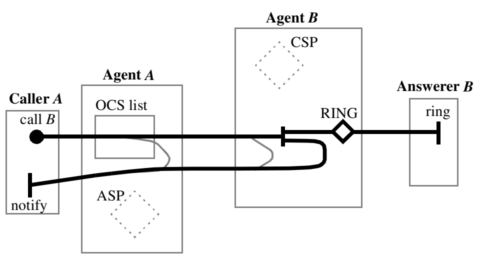
A Telephony-Agency UCM



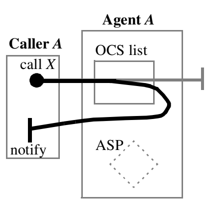
UCM Notation



Selected Scenarios

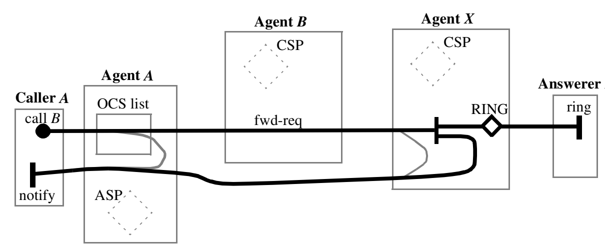


Path view of "*A* connects to *B*" scenario.

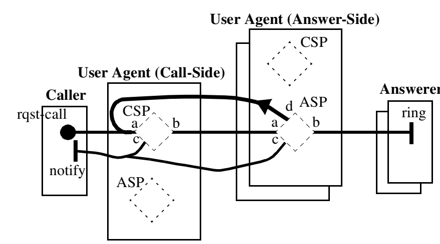


Path view of "*A* refuses call to *B*" scenario.

OCS-CF Feature Interaction and its Resolution

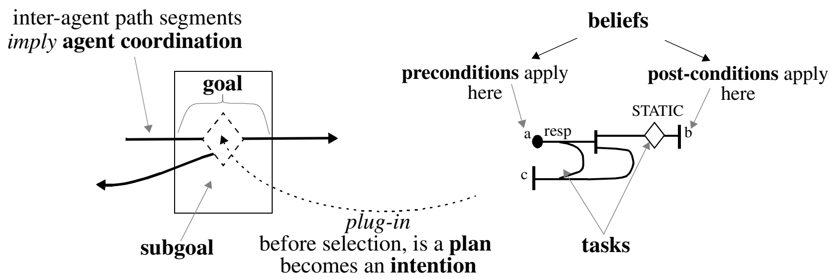


Feature interaction: A *allowed* to call X.



Feature interaction resolution with a final UCM

From UCMs to Agent Metamodels



Relationships are general ones applicable to agent metamodels that may differ in detail

Summary From the Top Down

- Constructive approach.
- UCMs used for visual 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 and 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 seems 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.
- Local salience is too simple. Extensions with OPI are planned for a more flexible and enhanced control of dynamic situations at run time.

Work in Progress (1)

1) OPI Model (Obligation-Permissions-Interdiction)

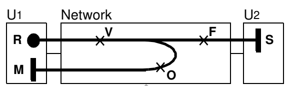
- Structure goals in a hierarchy of alternative, parallel, and sequential sub-goals.
- Based on deontic logic with formal propagation rules for its modalities (*O*, *P*, and *I*) and for acceptability.
- Reasoning and determination of acceptability in the presence of conflicting goals (*costs* attached to modalities).
- Usable directly as messages for proposal (intentions).
- Agents involved in a negotiation may have different hierarchies (subscribed features, policies).
- Linkable to UCMs?

Work in Progress (2)

2) From UCMs to LOTOS

- Integration of features at the UCM level, and avoidance of trivial and artificial interactions.
- Synthesis of a LOTOS specification from the UCMs.
- Detection of remaining interactions through testing.

The structure is mapped onto a set of processes composed through channels or shared events.



Each component becomes a process that implements all the paths that cross it (possibly from multiple scenarios).

```
specification   Connection[R,V,F,S,O,M] :   noexit
(* Abstract Data Types *)
behaviour
   hide  Chan1, Chan2   in
      ( U1[R,M,Chan1] ||| U2[S,Chan2]
      |[ Chan1, Chan2 ]|
      Network[V,F,O,Chan1, Chan2]
where
   (* Component processes ... *)
endspec   (* Connection *)
```

```
process  Network[V,F,O,Chan1, Chan2] :   noexit  :=
   Chan1 !U1ToNetwork ! m1;  V;
   (
      F; Chan2 !NetworkToU2 ! m2;
      Network[V,F,O,Chan1, Chan2]
      []
      O; Chan1 !NetworkToU1 ! m3;
      Network[V,F,O,Chan1, Chan2]
   )
endproc   (* Network *)
```


Feature-Interaction Visualization and Resolution in an Agent Environment

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.
- Potentially usable for upcoming new types of features.