On Coordination Languages and Models

Jean-Marie Jacquet

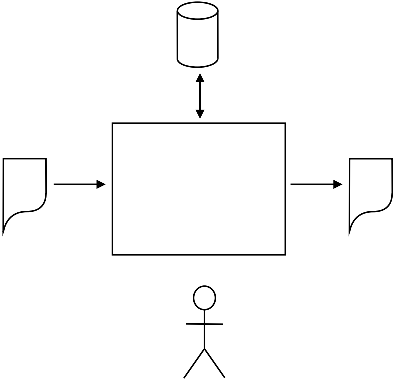
Faculty of Computer Science
University of Namur, Belgium

Academic year 2017–2018

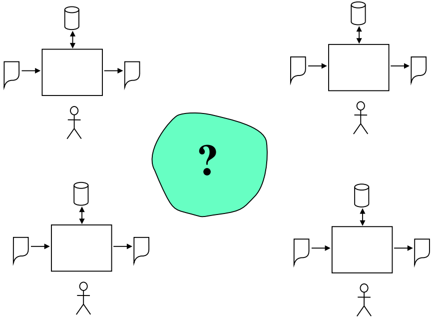
Outline

Outline

Classical systems



Modern systems

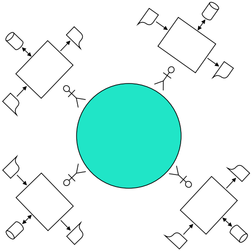


Outline

Different points of view

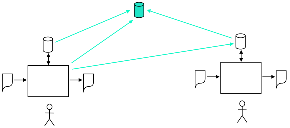
- Organizations
- Databases
- Messages
- Processes

Organizations



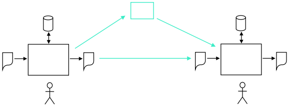
- organization structure
- agents
- roles
- desires

Databases



- DB structure
- DB integration
- code generation
- wrapper specifications and coding

Messages



- message syntax
- message semantics
- message coordination

Script langages

- Bourne shell (Unix)

```
ls -al > out.txt
mpage -P file.ps | lp -d folon
last | awk '{print $1}' | sort -u | rsh backus expand | awk '{print $1}
```

Script langages (cont'd)

- Manifold

```
process p:                    process q:

  compute m1                    ...
  send m1 to q                  receive m1
  compute m2                    let z be the sender of m1
  send m2 to q                  receive m2
  ...                           compute m using m1 and m2
  receive m                     send m to z
  ...                           ...
```

```
process p:                process q:                process c:

  compute m1                read m1 from              create channel
  write m1 to                 input port i1             p.o1 -> q.i1
    output port o1          read m2 from              create channel
  compute m2                  input port i2             p.o2 -> q.i2
  write m2 to               compute m                 create channel
    output port o2            using m1 and m2           q.o1 -> p.i1
  ...                       write m to
  read m from                 output port o1
    input port i1
  ...
```

KQML

- Agent communication language
- Darpa initiative (1993)
- Two components
  - KQML: Knowledge Query and Manipulation Language
  - KIF: Knowledge Interchange Format

KIF

- Caracterization
  - language for expressing message contents
  - based on first-order language
  - LISP syntax
- Express
  - object properties: “Michael is vegetarian”
  - object relationships: “Ann and Michael are married”
  - general propreties: “Every man has a mother”

KIF (cont'd)

- Concretely, KIF proposes
  - classical logic operators: `and`, `or`, `not`, `forall`, `exists`
  - base language: numbers, characters, strings, ...
  - relationship between objects: $\leq$, $+$, ...
  - LISP notation for manipulating objects

Examples

- The temperature of m_1 is 83 Celsius degrees

```
(= (temperature m1) (scalar 83 Celsius))
```

- A bachelor is a non married man

```
(defrelation bachelor (?x) :=
  (and (man ?x)
    (not (married ?x))))
```

- Anyone who is a person is also a mammal

```
(defrelation person (?x) :=> (mammal ?x))
```

KQML

- KQML = language for defining communication actions
  - ask-if
  - perform
  - tell
  - reply

KQML (cont'd)

- any action is specified by different attributes
  - `content`: message content
  - `reply-with`: identifier for answering
  - `in-reply-to`: reference for an answer
  - `sender`: message sender
  - `receiver`: message receiver
  - `language`: language in which the message content is expressed
  - `ontology`: definition of the terminology

Example

```
(ask-one
   :content (PRICE IBM ?price)
   :receiver stock-server
   :language LPROLOG
   :ontology NYSE-TICKS
)
```

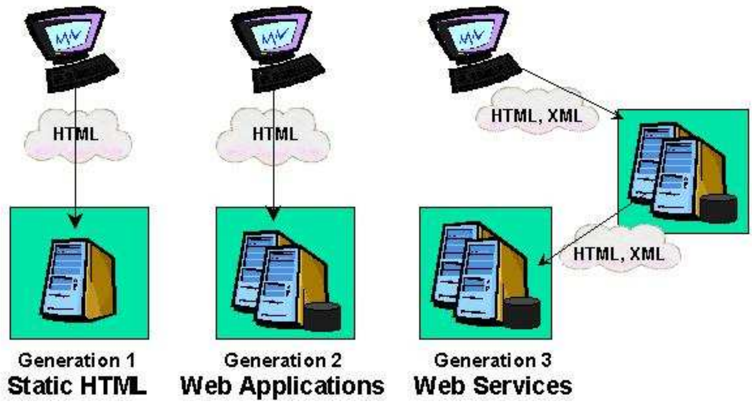
Dialogue example

```
(evaluate                    :sender A
  :receiver B                :language KIF
  :ontology motors           :reply-with q1
  :content (val (torque m1)))

(reply                       :sender B
  :receiver A                :language KIF
  :ontology motors           :in-reply-to q1
  :content (= (torque m1) (scalar 12 kgf)))
```

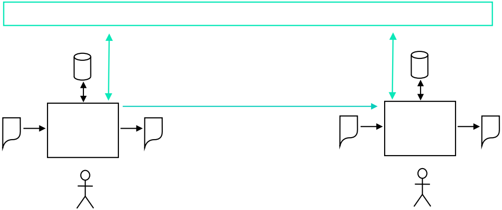
Example 2

```
(stream-about
  :sender A               :receiver B
  :language KIF           :ontology motors
  :reply-with q1          :content m1)

(tell                     :sender B
  :receiver A             :in-reply-to q1
  :content (= (torque m1) (scalar 12kgf)))

(tell                     :sender B
  :receiver A             :in-reply-to q1
  :content (= (status m1) normal))

(eos                      :sender B
  :receiver A             :in-reply-to q1)
```

Web services



Processes



RMI, COM, DCOM, CORBA, ToolBus

Outline

Definition

The most accepted view of coordination is that of managing the interaction and dependencies between the entities of a system – whether they are agents, processes, molecules or individuals.

[Omicini, Zambonelli, Klusch, Tolksdorf, 2001]

Programming paradigms

1950s Machine langages

1960s Imperative programming

$\Rightarrow$ granularity and abstraction of actions

1970s Structured programming

$\Rightarrow$ programming methodologies

1980s Declarative programming/object-oriented programming

$\Rightarrow$ specification and interaction

Programming paradigms (cont'd)

1990s Component-based programming

$\Rightarrow$ (non sequential) patterns of interaction

2010s Reactive programming/Contextual programming

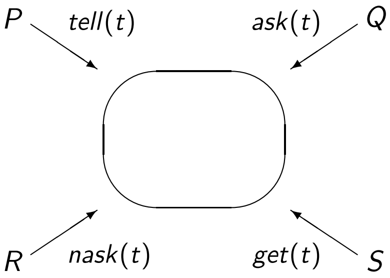
$\Rightarrow$ adaptation to new contexts

Objectives of the lecture

- expose work done in the CoordiNam laboratory
- expose basic mechanisms of coordination models and languages
- project : study through an implementation in Scala

Outline

Snapshot



$$\begin{array}{lcl} C & ::= & tell(t) \mid ask(t) \mid nask(t) \mid get(t) \\ A & ::= & C \mid A \,;\, A \mid A \,||\, A \mid A + A \end{array}$$

Transition system (1)

$$< tell(t), \sigma > \longrightarrow < E, \sigma \cup \{t\} >$$

$$< ask(t), \sigma \cup \{t\} > \longrightarrow < E, \sigma \cup \{t\} >$$

$$\frac{t \notin \sigma}{< nask(t), \sigma > \longrightarrow < E, \sigma >}$$

$$< get(t), \sigma \cup \{t\} > \longrightarrow < E, \sigma >$$

Transition system (2)

$$\frac{<A, \sigma> \longrightarrow <A', \sigma'>}{<A \;;\; B, \sigma> \longrightarrow <A' \;;\; B, \sigma'>}$$

$$\frac{<A, \sigma> \longrightarrow <A', \sigma'>}{\begin{array}{c}<A \,||\, B, \sigma> \longrightarrow <A' \,||\, B, \sigma'> \\ <B \,||\, A, \sigma> \longrightarrow <B \,||\, A', \sigma'>\end{array}}$$

$$\frac{<A, \sigma> \longrightarrow <A', \sigma'>}{\begin{array}{c}<A + B, \sigma> \longrightarrow <A', \sigma'> \\ <B + A, \sigma> \longrightarrow <A', \sigma'>\end{array}}$$

Properties

- persistent broadcast communication ($><$ channel communication)
- associative memory
- clear separation between communication and computation

  $\Rightarrow$ "divide and conquer" methodology

  - code program fragments assuming that required data will eventually be available
  - compose these fragments by establishing that data is indeed provided.

  $\Rightarrow$ "coordination langage": allow for the composition of programs written in different languages

A restaurant room

$$\begin{array}{rcl} room & = & producer \mid\mid harry \mid\mid peter \\ producer & = & prepare(Item); tell(Item); producer \\ harry & = & get(ice); harry + get(cake); harry \\ peter & = & get(water); peter + get(wine); peter \end{array}$$

Questions

1. $harry \stackrel{?}{=} harry'$

$$
\begin{aligned}
harry &= get(ice); harry + get(cake); harry \\
harry' &= (get(ice) + get(cake)); harry
\end{aligned}
$$

2. $paul \stackrel{?}{=} paul'$

$$
\begin{aligned}
paul &= get(water); get(bread); paul \\
&\quad + get(water); get(cheese); paul \\
paul' &= get(water); (get(bread) + get(cheese)); paul'
\end{aligned}
$$

Example 2: a tourist

$$
\begin{array}{rcl}
W\_station\_bxl & = & local\ x, y\ in \\
 & & \quad compute\_weather(namur, x, y)\ ;\ tell(\langle namur, x, y \rangle) \\
 & & end \\
W\_station\_lis & = & local\ w\ in \\
 & & \quad compute\_sky(funchal, w)\ ;\ tell(\langle funchal, w \rangle) \\
 & & end \\
Tourist & = & local\ min, max\ in \\
 & & \quad ask(\langle namur, ?min, ?max \rangle)\ ;\ drive\_to(namur) \\
 & & \quad + \\
 & & \quad ask(\langle funchal, sunny \rangle)\ ;\ fly\_to(funchal) \\
 & & end
\end{array}
$$

Processes as active data

Data

tellt(t), gett(t), askt(t), naskt(t)

Processes

tellp(t), getp(t), askp(t), naskp(t)

Flight reservation system

- Problem description
  - four flights: `ba023`, `sn720`, `nw129`, `kl283`
  - questioned by three terminals (identified as 1, 2 and 3)
  - by means of `reservation` messages.
- Code in Prolog style

```
flight_syst :-
    tellp(terminal(1)), tellp(terminal(2)), tellp(terminal(3)),
    tellp(flight(ba023,80)), tellp(flight(sn720,150)),
    tellp(flight(nw129,68)), tellp(flight(kl283,185)).
```

Code (cont'd)

```
terminal(Id) :-
        user_input(Id,Flight,SeatsRequested),
        tellt(reservation(Flight,SeatsRequested)),
        gett(acknowledge(Flight,SeatsRequested,SeatsLeft,Status)),
        user_output(Id,SeatsLeft,Status).

flight(Name, SeatsAvailable) :-
        gett(reservation(Name,SeatsRequested)),
        reserve_seats(SeatsRequested,SeatsAvailable,SeatsLeft,Status),
        tellt(acknowledge(Name,SeatsRequested,SeatsLeft,Status),
        flight(Name, SeatsLeft).

reserve_seats(SeatsRequested,SeatsAvailable,SeatsLeft,accepted) :-
        SeatsAvailable >= SeatsRequested,
        SeatsLeft is SeatsAvailable - SeatsRequested.
reserve_seats(SeatsRequested,SeatsAvailable,SeatsAvailable,refused) :-
        SeatsAvailable < SeatsRequested.
```

Enhanced matching

Communication variables

- keep the communication through the tuple space
- use special variables to input/output data to the "host" program

Default matching

- use a left-to-right parsing
- a value matches that same value
- a communication variable matches any value

Psi-terms

- use pairs of item name – value
- perform the matching according to the item name

Formal definition (1)

ψ-term

Construct of the form $f(a_1 = v_1, \cdots, a_m = v_m)$ where

- f/n is a functor such that $m \leq n$
- the a_i's are distinct constants
- v_i denotes an integer, a string of characters, a ψ-term or a communication variable
- any communication variable appears at most once

Closed *psi*-term

A ψ-term is said to be closed if it contains no communication variable.

Formal definition (2)

Correspondance

$\psi_1 = f(a_1 = v_1, \cdots, a_l = v_l)$ corresponds to
$\psi_2 = f'(a'_1 = v'_1, \cdots, a'_m = v'_m)$ iff

1. f and f' are identical functors with same arities;
2. $\{a_i : 1 \leq i \leq l\} \subseteq \{a'_j : 1 \leq j \leq m\}$;
3. for any i such that v_i is an integer or a string of characters, if $a_i = a'_j$ then $v_i = v'_j$;
4. for any i such that v_i is a ψ-term, if $a_i = a'_j$ then v_i corresponds to v'_j.

Formal definition (3)

Binding

$$\theta : Scvar \rightarrow (N \cup Sstring \cup Scpterm \cup \{\perp\})$$

Matching

Let $\psi_1 = f(a_1 = v_1, \cdots, a_l = v_l)$, $\psi_2 = f'(a'_1 = v'_1, \cdots, a'_m = v'_m)$.
Assume ψ_2 is closed.
ψ_1 matches ψ_2 iff $\psi_1\theta$ corresponds to ψ_2, for some binding θ.

Outline

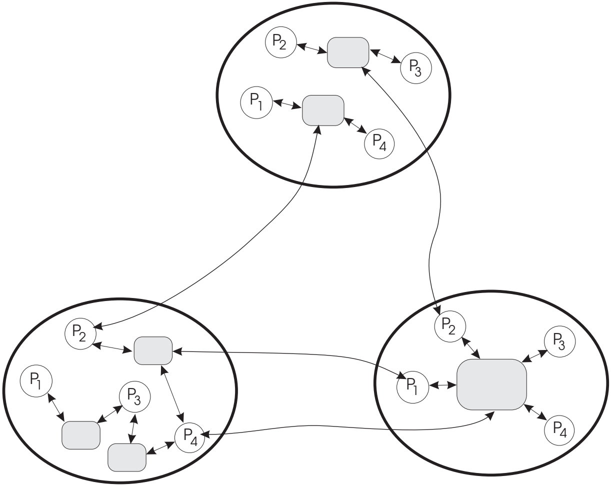
Approaches to distribution

- *Classical approach:*
  - distribute a single entity (eg the dataspace)
  - use a manager to coordinate the pieces
  - think of the entity as a non-fractioned object
- *Our approach:*
  - coordinate several applications
  - avoid the use of a master manager

Four steps

1. introduce multiple blackboards,
2. distribute blackboards on computing resources,
3. introduce aliases allowing access to non-local blackboards,
4. perform load balancing by moving blackboards between locations.

Distribution in Bach



Multiple blackboards

- **Concept**

  $$Blackboard = dataspace + processes$$

- **The Bach approach**
  - blackboards manipulated by *tell*, *get*, *ask*, *nask* primitives,
  - processes can access data and processes on any blackboard.

Distribution

- **Concepts**
  - A *processor* consists of computing resources.
  - An *application* consists of the executions of several blackboards launched by a common initial instruction.
  - An *abstract machine* consists in a pair processor-application.
- **The Bach approach**
  - blackboards are created locally;
  - an execution in Bach is composed of the concurrent executions of applications;
  - the execution of a blackboard is made with respect to a program attached to the application to which the blackboard belongs;
  - access to nonlocal blackboards takes place via special links.

Aliases

- **Concepts**

  For the purposes of dynamic evolution, an extra indirection on the blackboard naming is desirable

- **The Bach approach**

  Introduce *virtual blackboards* as special blackboards containing no data and no processes but pointing to other (possibly virtual) blackboards.

Load balacing

- **Concepts**

  To dynamically balance the load of the computations, parts of executions may need to migrate from one place to another.

- **The Bach approach**

  Introduce primitives to exchange blackboards and to relink them.

Language: communication primitives

$tellbb(bbn, bbt, bbp)$	$tellt(bbn, t)$	$tellp(bbn, p)$
$getbb(bbn)$	$gett(bbn, t)$	$getp(bbn, p)$
$askbb(bbn)$	$askt(bbn, t)$	$askp(bbn, p)$
$naskbb(bbn)$	$naskt(bbn, t)$	$naskp(bbn, p)$

$tellvb(bbn_1, bbn_2@ma_2)$
$relink(bbn_1, bbn_2@ma_2)$,
$exch(bbn_1, bbn_2@ma_2)$

Auxiliary concepts

- **Processes:** $(\Leftarrow G \Diamond \theta)$
- **Contexts:**
  - ∇ is a context s.t. $\nabla[\![A]\!] = A$
  - If c is a context and if A is an agent, then

    $$c \, ; A \qquad c \parallel A \qquad A \parallel c$$

    are contexts s.t.

    $$\begin{array}{rcl} (c \, ; A)[\![A']\!] & = & c[\![A']\!] \, ; A \\ (c \parallel A)[\![A']\!] & = & c[\![A']\!] \parallel A \\ (A \parallel c)[\![A']\!] & = & G \parallel c[\![G']\!] \end{array}$$

- **Configurations:** sets of elements of the form
  $\langle bbn@ma, bbt, bbp \rangle$. (real blackboard)
  $\langle bbn_1@ma_1 \rightsquigarrow bbn_2@ma_2 \rangle$. (virtual blackboard)

Tell reductions: Real blackboard

$$(\mathrm{T}_b) \qquad \{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![tellbb(bbn, bbt, bbp)]\!]\Diamond\theta]\rangle\ |\!\} \rightarrow$$

$$\{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\Box]\!]\Diamond\theta]\rangle\ |\!\} \cup \{\langle bbn@ma, bbt\theta, bbg'\rangle\}$$

$$if \left\{ \begin{array}{l} \text{no blackboard is identified by } bbn@ma \text{ in the initial} \\ \qquad \text{configuration} \\ bbt\theta \text{ is composed of closed } \psi\text{-terms} \\ bbg' \text{ is obtained from } bbg\theta \\ \qquad \text{by freshly renaming the communication variables} \end{array} \right\}$$

Tell reduction: virtual blackboard

$$(\mathrm{T}_v) \qquad \{\!\!|\ \langle n@ma, bt, m[\Leftarrow c[\![tellvb(bbn_1, bbn_2@ma_2)]\!]\Diamond\theta]\rangle\ |\!\!\} \rightarrow$$

$$\{\!\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\Box]\!]\Diamond\theta]\rangle\ \langle bbn_1@ma \rightsquigarrow bbn_2@ma_2\rangle\ |\!\!\}$$

$$\mathit{if} \left\{ \begin{array}{l} \text{there exists a blackboard in the initial} \\ \text{configuration identified by } bbn_2@ma_2 \end{array} \right\}$$

Tell reduction: term

$$
(\mathrm{T}_t) \qquad \begin{array}{l} \{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![tellt(bbn,t)]\!]\Diamond\theta]\rangle\ \langle bbn_0@ma_0 \rightsquigarrow bbn_1@ma_1\rangle\ \cdots \\ \quad \langle bbn_{i-1}@ma_{i-1} \rightsquigarrow bbn_i@ma_i\rangle\ \langle bbn_i@ma_i, bt', bp'\rangle\ |\!\} \rightarrow \\ \\ \quad \{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\triangle]\!]\Diamond\theta]\rangle\ \langle bbn_0@ma_0 \rightsquigarrow bbn_1@ma_1\rangle\ \cdots \\ \quad \langle bbn_{i-1}@ma_{i-1} \rightsquigarrow bbn_i@ma_i\rangle\ \langle bbn_i@ma_i, bt' + \{u\}, bp'\rangle\ |\!\} \end{array}
$$

$$
\mathit{if} \left\{ \begin{array}{l} u = t\theta \text{ is closed} \\ bbn_0 = bbn \\ 0 \leq i \end{array} \right\}
$$

Get reductions: virtual blackboard

$$(\mathsf{G}_v) \qquad \{\!\!|\ \langle n@ma, bt, m[\Leftarrow c[\![getbb(bbn)]\!]\Diamond\theta]\rangle\ \langle bbn@ma \rightsquigarrow bbn'@ma'\rangle\ |\!\!\} \rightarrow$$

$$\{\!\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\Box]\!]\Diamond\theta]\rangle\ |\!\!\}$$

Exchange reduction

(E$_1$)
$$\{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![exch(bbn_1, bbn_2@ma_2)]\!]\Diamond\theta]\rangle\ \langle bbn_1@ma, bbt, bbp\rangle\ \langle bbn_2@ma_2 \rightsquigarrow bbn_1@ma\rangle\ |\!\} \rightarrow$$
$$\{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\Box]\!]\Diamond\theta]\rangle\ \langle bbn_1@ma \rightsquigarrow bbn_2@ma_2\rangle\ \langle bbn_2@ma_2, bbt, bbp\rangle\ |\!\}$$

(E$_2$)
$$\{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![exch(bbn_1, bbn_2@ma_2)]\!]\Diamond\theta]\rangle\ \langle bbn_1@ma \rightsquigarrow bbn_3@ma_3\rangle\ \langle bbn_2@ma_2 \rightsquigarrow bbn_1@ma\rangle\ |\!\} \rightarrow$$
$$\{\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\Box]\!]\Diamond\theta]\rangle\ \langle bbn_1@ma \rightsquigarrow bbn_2@ma_2\rangle\ \langle bbn_2@ma_2 \rightsquigarrow bbn_3@ma_3\rangle\ |\!\}$$

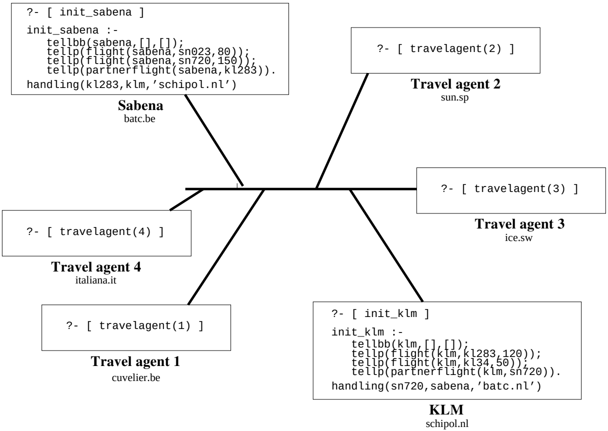
Relink reduction

$$\text{(Rel)} \qquad \{\!\!|\ \langle n@ma, bt, m[\Leftarrow c[\![relink(bbn_1, bbn_3@ma_3)]\!]\lozenge\theta]\rangle \ \langle bbn_1@ma \rightsquigarrow bbn_2@ma_2\rangle \ |\!\!\}$$

$$\rightarrow \{\!\!|\ \langle n@ma, bt, m[\Leftarrow c[\![\Box]\!]\lozenge\theta]\rangle \ \langle bbn_1@ma \rightsquigarrow bbn_3@ma_3\rangle \ |\!\!\}$$

$$\textit{if} \left\{ \begin{array}{l} \text{there exists a blackboard in the initial} \\ \text{configuration identified by } bbn_3@ma_3 \end{array} \right\}$$

Coordinating travel agencies



Travel agent

```
travelagent(Id) :-
   user_input(Flight,SeatsRequested),
   carrier(Flight,Company,Machine),
   tellvb(vbb,Company@Machine),
   tellt(vbb,reservation(Flight,SeatsRequested,Id)),
   gett(vbb,acknowledge(Id,SeatsLeft,Status)),
   getbb(vbb),
   user_output(SeatsLeft,Status),
   travelagent(Id).

carrier(sn023,sabena,'batc.be').
carrier(sn720,sabena,'batc.be').
carrier(nw29,northwest,'jfk.com').
carrier(kl283,sabena,'batc.be').
```

Airline company

```
flight(Company, Flight, SeatsAvailable) :-
   gett(Company, reservation(Flight,SeatsReq,Id)),
   reserve_seats(SeatsReq,SeatsAvailable,SeatsLeft,Status),
   tellt(Company, acknowledge(Id,SeatsLeft,Status)),
   flight(Company, Flight, SeatsLeft).

partnerflight(Company, Flight) :-
   gett(Company, reservation(Flight,SeatsReq,Id)),
   handling(Flight,RealCompany,Machine),
   tellvb(vpartner,RealCompany@Machine),
   tellt(vpartner,reservation(Flight,SeatsReq,Id)),
   gett(vpartner,acknowledge(Id,SeatsLeft,Status)),
   getbb(vpartner),
   tellt(Company, acknowledge(Id,SeatsLeft,Status)),
   partnerflight(Company, Flight).
```

Outline

Relating blackboards: basic ideas

- *In* **primitive**

  $in(bbn, O)$: the object O is on the blackboard *bbn*

- **R-rules**

$$in(b_1, O_1), \cdots, in(b_j, O_j) \longrightarrow$$
$$in(b_{j+1}, O_{j+1}), \cdots, in(b_m, O_m)$$

  The presence of O_1, ..., O_j on blackboards b_1, ..., b_j implies the presence of O_{j+1}, ..., O_m on blackboards b_{j+1}, ..., b_m, respectively.

Variants

- some objects need to be consumed to produce others
- this production may actually lead to the creation of new objects on the blackboards
- the selection of objects may be enhanced in two ways:
  - by using conditions to strengthen the selection of objects based on unification only in the *in* primitive,
  - by, if need be, suffixing the *in* primitive by 't' and 'p' to explicitly distinguish between data and processes

General form

$$[In(b_1, O_1) : C_1, \cdots, In(b_i, O_i) : C_i] \; \{In(b_{i+1}, O_{i+1}) : C_{i+1}, \cdots, In(b_j, O_j) : C_j\}$$
$$\longrightarrow [In(b_{j+1}, O_{j+1}) : C_{j+1}, \cdots, In(b_k, O_k) : C_k] \; \{In(b_{k+1}, O_{k+1}) : C_{k+1}, \cdots, In(b_m, O_m) : C_m\}$$

where

- the *In*'s there stay either for *in*, *int*, or *inp*
- the *C*'s represent conditions of atoms defined by Horn clauses
- the square brackets and curly brackets are respectively used to indicate modifications and no modifications.

Design decisions

- Any condition C_k $(1 \leq k \leq j)$ may involve other variables than those of O_k under the restriction that the variables used in $C_1, \ldots, C_j$ are limited to those of $O_1, \ldots, O_j$.
- Any condition C_k, $j+1 \leq k \leq l$ should include as only variables those of $O_1, \ldots, O_j$ and of O_k.
- Only those objects O_k $(1 \leq k \leq j)$ physically present on b_k are considered to produce new objects.
- Objects written are understood as being duplicated.
- The constructive telling is operated incrementally each time a new object O_k is inserted on b_k $(1 \leq k \leq j)$;
- At the activation time of an r-rule, the objects O_k that are already present are also subject to the constructive telling and therefore also induce objects.

General relations

$$Rel \equiv \{R_1, \cdots, R_m\}$$

Examples

- **The forward relation**

$$forward(b_1, b_2) \equiv \{\ [in(b_1, X)]\ \{\}\ \longrightarrow [in(b_2, X)]\ \{\}\ \}$$

- **Inheritance relation**

$$inherit(b_1, b_2) \equiv \{\ []\ \{in(b_2, X)\}\ \longrightarrow []\ \{in(b_1, X)\}\ \}$$

Examples

- **Merge relation**

$$merge(b_1, b_2, b) \equiv \left\{ \begin{array}{l} [in(b_1, X)]\ \{\} \longrightarrow [in(b, X)]\ \{\} \\ [in(b_2, X)]\ \{\} \longrightarrow [in(b, X)]\ \{\} \end{array} \right\}$$

- **The duplicate relation**

$$duplicate(b, b_1, b_2) \equiv \{\ [in(b, X)]\ \{\} \longrightarrow [in(b_1, X), in(b_2, X)]\ \{\}\ \}$$

New primitives

- $telllink(bbn, rn)$
- $getlink(bbn, rn)$
- $asklink(bbn, rn)$
- $nasklink(bbn, rn)$

Outline

Why time?

- **Application need**
  - Request on the Web to be satisfied in a reasonable amount of time
  - Request for an ambulance to be answered within a critical amount of time
- **The coordination community**
  - tcc, tdcc (Saraswat, Jagadeesan, Gupta, 1994, 1996)
  - tccl (De Boer, Gabbrielli, Meo, 2000)
  - Oz (Smolka, 1995)
  - JavaSpaces (Freeman, Hupfer, Arnold, 1999)

Hypothesis

Use the two-phase functioning approach

- First phase: elementary actions of statements are executed.
  - actions are supposed to take no time
  - composition operators are supposed to take no time
- Second phase: time progresses by one unit
  - when no actions can be reduced or when all the components encounter a special timed action

Example

$$
\begin{array}{rcl}
W\_Station\_lis & = & local\ \ w\ \ in \\
 & & \quad delay(10)\ ;\ compute\_sky(funchal, w)\ ; \\
 & & \quad tell_{10}(\langle funchal, w\rangle)\ ;\ W\_Station\_lis \\
 & & end
\end{array}
$$

$$
\begin{array}{rcl}
Tourist & = & local\ \ min, max\ \ in \\
 & & \quad ask_5(\langle funchal, sunny\rangle)\ ;\ fly\_to(funchal) \\
 & & \quad + \\
 & & \quad delay(5)\ ;\ drive\_to(namur) \\
 & & end
\end{array}
$$

The $\mathcal{D}$ family (1)

Syntax

$$C \quad ::= \quad tell(t) \mid ask(t) \mid get(t) \mid nask(t) \mid delay(d)$$

Semantics

$$(D1) \qquad \frac{A \neq E, A \neq A^{-}, <A, \sigma> \not\to}{<A, \sigma> \rightsquigarrow <A^{-}, \sigma>}$$

$$(D2) \qquad <delay(0), \sigma> \longrightarrow <E, \sigma>$$

The $\mathcal{D}$ family (2)

$$
\begin{array}{rcl}
tell(t)^- & = & tell(t) \\
ask(t)^- & = & ask(t) \\
nask(t)^- & = & nask(t) \\
get(t)^- & = & get(t) \\
delay(0)^- & = & delay(0) \\
delay(d)^- & = & delay(d-1) \\
(B\ ;\ C)^- & = & B^-\ ;\ C \\
(B \mid\mid C)^- & = & B^- \mid\mid C^- \\
(B\ +\ C)^- & = & B^-\ +\ C^-
\end{array}
$$

The $\mathcal{R}$ family (1)

Syntax

$$C \quad ::= \quad tell_d(t) \mid ask_d(t) \mid get_d(t) \mid nask_d(t)$$

Semantics

$$(T0) \qquad <tell_0(t), \sigma> \longrightarrow <E, \sigma>$$

$$(Tr) \qquad \frac{d > 0}{<tell_d(t), \sigma> \longrightarrow <E, \sigma \cup \{t_d\}>}$$

$$(Ar) \qquad \frac{d > 0}{<ask_d(t), \sigma \cup \{t_k\}> \longrightarrow <E, \sigma \cup \{t_k\}>}$$

The $\mathcal{R}$ family (2)

Semantics

$$(Nr) \qquad \frac{d > 0,\ \nexists k : t_k \in \sigma}{<nask_d(t), \sigma> \longrightarrow <E, \sigma>}$$

$$(Gr) \qquad \frac{d > 0}{<get_d(t), \sigma \cup \{t_k\}> \longrightarrow <E, \sigma>}$$

$$(Wr) \qquad \frac{A \neq E, A \neq A^{-} \text{ or } \sigma \neq \sigma^{-}, <A, \sigma> \not\rightarrow}{<A, \sigma> \leadsto <A^{-}, \sigma^{-}>}$$

The $\mathcal{R}$ family (2)

$$
\begin{array}{rcl}
tell_d(t)^- & = & tell_d(t) \\
ask_d(t)^- & = & ask_{max\{0,d-1\}}(t) \\
nask_d(t)^- & = & nask_{max\{0,d-1\}}(t) \\
get_d(t)^- & = & get_{max\{0,d-1\}}(t) \\
E^- & = & E \\
(B\ ;\ C)^- & = & B^-\ ;\ C \\
(B\ ||\ C)^- & = & B^-\ ||\ C^- \\
(B\ +\ C)^- & = & B^-\ +\ C^-
\end{array}
$$

$$\sigma^- = \{t_{d-1} : t_d \in \sigma, d > 1\}$$

The $\mathcal{W}$ family

Syntax

$$C \quad ::= \quad tell(t) \mid ask(t) \mid get(t) \mid nask(t) \mid wait(m)$$

Semantics

$$(W1) \qquad \frac{A \neq E,\, A \gg u,\, <A,\sigma>_u \not\rightarrow}{<A,\sigma>_u \rightsquigarrow <A,\sigma>_{u+1}}$$

$$(W2) \qquad \frac{u \geq v}{<wait(v),\sigma>_u \longrightarrow <E,\sigma>_u}$$

$A \gg u$ iff A contains a $wait(m)$ with $m > u$

The $\mathcal{I}$ family (1)

Syntax

$$C \quad ::= \quad tell_{[b:e]}(t) \mid ask_{[b:e]}(t) \mid get_{[b:e]}(t) \mid nask_{[b:e]}(t)$$

Semantics

$$(Ta) \qquad \frac{b \leq u \leq e}{<tell_{[b:e]}(t), \sigma>_u \longrightarrow <E, \sigma \cup \{t_{[u:e]}\}>_u}$$

$$(Aa) \qquad \frac{b \leq u \leq e, b' \leq u \leq e'}{\begin{array}{c} <ask_{[b:e]}(t), \sigma \cup \{t_{[b':e']}\}>_u \\ \longrightarrow \;\; <E, \sigma \cup \{t_{[b':e']}\}>_u \end{array}}$$

The $\mathcal{I}$ family (2)

Semantics

$$(Na) \qquad \frac{b \leq u \leq e, \quad \not\exists b', e' : b' \leq u \leq e' \wedge t_{[b':e']} \in \sigma}{<nask_{[b:e]}(t), \sigma>_u \longrightarrow <E, \sigma>_u}$$

$$(Ga) \qquad \frac{b \leq u \leq e, b' \leq u \leq e'}{<get_{[b:e]}(t), \sigma \cup \{t_{[b':e']}\}>_u \longrightarrow <E, \sigma>}$$

$$(Wa) \qquad \frac{A \neq E, A \gg u \text{ or } \sigma \gg u, <A, \sigma> \not\rightarrow}{<A, \sigma>_u \rightsquigarrow <A, \sigma^{+u}>_{u+1}}$$

The $\mathcal{I}$ family (3)

$$\sigma \gg u \text{ iff } \exists t_{[b:e]} \in \sigma : (e \neq \infty \wedge e > u)$$

$$\sigma^{+u} = \{t_{[max\{b,u+1\}:e]} : t_{[b:e]} \in \sigma, u + 1 \leq e\}.$$

Example: delayed requests

- Abstracting the web as a tuple space, a request for information on the web may be typically programmed as follows:

$$\begin{array}{l} tell_d(request)\ ; \\ \quad (get_d(answer) + (delay(d)\ ;\ tell_\infty(no\_answer))) \end{array}$$

- Similarly, exceptions after some delays may be programmed as

$$\begin{array}{rcl} ask_d(t)\Box A & \equiv & ask_d(t) + (delay(d)\ ;\ A) \\ get_d(t)\Box A & \equiv & get_d(t) + (delay(d)\ ;\ A) \\ nask_d(t)\Box A & \equiv & nask_d(t) + (delay(d)\ ;\ A) \end{array}$$

Example: Do . . . watch construct

$$do\ tell_d(t)\ watch\ e\ cont\ B \equiv$$

$$(ask_\infty(e)\ ;\ B) + (nask_\infty(e)\ ;\ tell_d(t))$$

$$do\ C_d(t)\ watch\ e\ cont\ B \equiv$$

$$\begin{cases} (ask_\infty(e)\ ;\ B) \\ \quad +(nask_\infty(e)\ ;\ C_1(t)\ ;\ do\ C_{d-1}(t)\ watch\ e\ cont\ B) \\ \quad \text{if } C \in \{ask, nask, get\} \text{ and } d > 1 \\ (ask_\infty(e)\ ;\ B) + (nask_\infty(e)\ ;\ C_d(t)) \\ \quad \text{if } C \in \{ask, nask, get\} \text{ and } d \leq 1 \end{cases}$$

$$do\ delay(d)\ watch\ e\ cont\ B \equiv$$

$$\begin{cases} (ask_\infty(e)\ ;\ B) \\ \quad +(nask_\infty(e)\ ;\ delay(1)\ ;\ do\ delay(d-1)\ watch\ e\ cont\ B) \\ \quad \text{if } d \geq 1 \\ (ask_\infty(e)\ ;\ B) + nask_\infty(e) \\ \quad \text{otherwise} \end{cases}$$

Example: Do . . . watch construct (2)

$$
\begin{aligned}
& do\ X\ ;\ Y\ watch\ e\ cont\ B \equiv \\
& \qquad do\ X\ watch\ e\ cont\ B\ ;\ do\ Y\ watch\ e\ cont\ B \\
& do\ X\ ||\ Y\ watch\ e\ cont\ B \equiv \\
& \qquad do\ X\ watch\ e\ cont\ B\ ||\ do\ Y\ watch\ e\ cont\ B \\
& do\ X\ +\ Y\ watch\ e\ cont\ B \equiv \\
& \qquad do\ X\ watch\ e\ cont\ B\ +\ do\ Y\ watch\ e\ cont\ B
\end{aligned}
$$

Outline

The untimed case

- Concurrency achieved by the threads library of Solaris
- Distribution achieved by RPC/socket
- Tuple space = token-indexed list of tokens
- For each token, we keep track of
  - number of occurrences of the token
  - a list of suspended processes
- Each token is protected by its own lock
- The implementation of the basic primitives is achieved as one might guess

Outline

Timed primitives

- Implement the primitives dealing with absolute time only
- A period of validity is associated with the tokens and processes of waiting lists
- Waking-up facilities provided by the operating system are used to
    - force wait primitives to succeed when the specified waiting time has been reached
    - force timed ask, nask, get to fail when their period of validity is over
    - remove tokens whole period of validity is over

Outline

Applications to MANETs

- **Application view**
  - everything occurs on the local blackboard
  - synchronization based on availability of information
  - contextual information available “by magic”
- **Middelware view**
  - Blackboard relations provide this magic
    - ! hosts may move out of connection
    - $\Rightarrow$ need for dynamic activation of relation
    - $\Rightarrow$ introduce events and reactions to them
  - ...

Applications to MANETs (2)

- **Middelware view (2)**
  - Timeouts
    - timed primitives allow to express timeouts
    - ! time considered with respect to local clock
    - ! get primitive handled with special protocol
  - Filter information and hosts
    - $\Rightarrow$ handle access rights and policy
  - Several relations may be used to satisfy a query
    - $\Rightarrow$ introduce priorities and make them vary in time

Events and reactions to events

- Special tuples written by dedicated processes

$$\langle connect, h\rangle,\ \langle disconnect, h\rangle,\ \langle quality, h, q\rangle$$

- Reaction

$$in(\langle connected, X\rangle) \ \Rightarrow \ tell([in(Y)@self] \longrightarrow_b [in(Y)@X])@X$$

Access rights and policies

- Special hidden attributes of tuples, blackboard relations, primitives
- inherited from the creating processes
- accessed if capabilities are matched