Timed Coordination Artifacts with ReSpecT

Mirko Viroli, Alessandro Ricci
DEIS, Sede di Cesena
Alma Mater Studiorum, Università di Bologna
{mviroli + aricci}@deis.unibo.it
properly filtered by Andrea Omicini

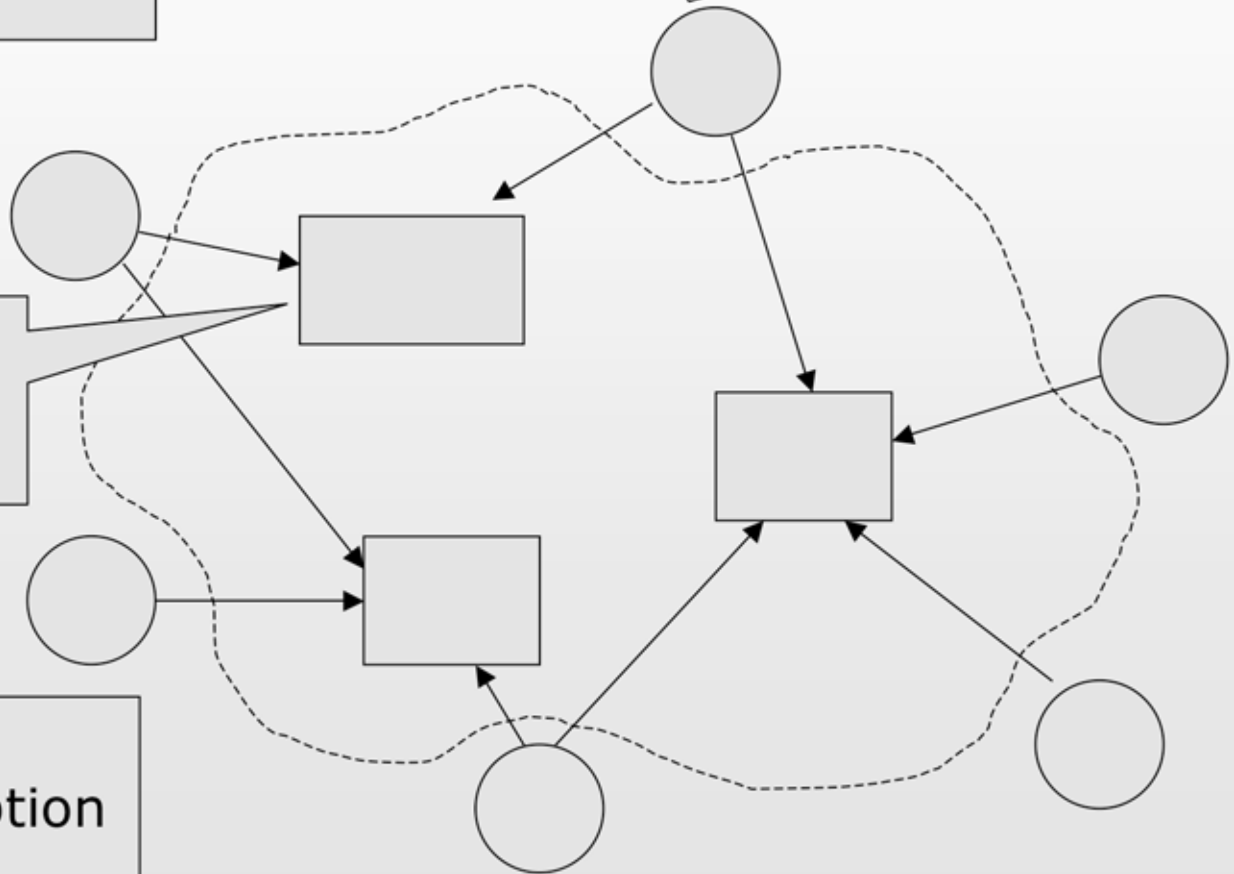
Outline

- General purpose coordination for MAS
- TuCSoN and the ReSpecT language
- Timed ReSpecT
- Examples of application
- Conclusions

TuCSoN Infrastructure

TuCSoN infrastructure: provides coordination services to agents in a distributed setting

Agents:
S/W components
Intelligent systems



Tuple Centre:
Programmable tuple space

Interactions:
production/consumption of tuples

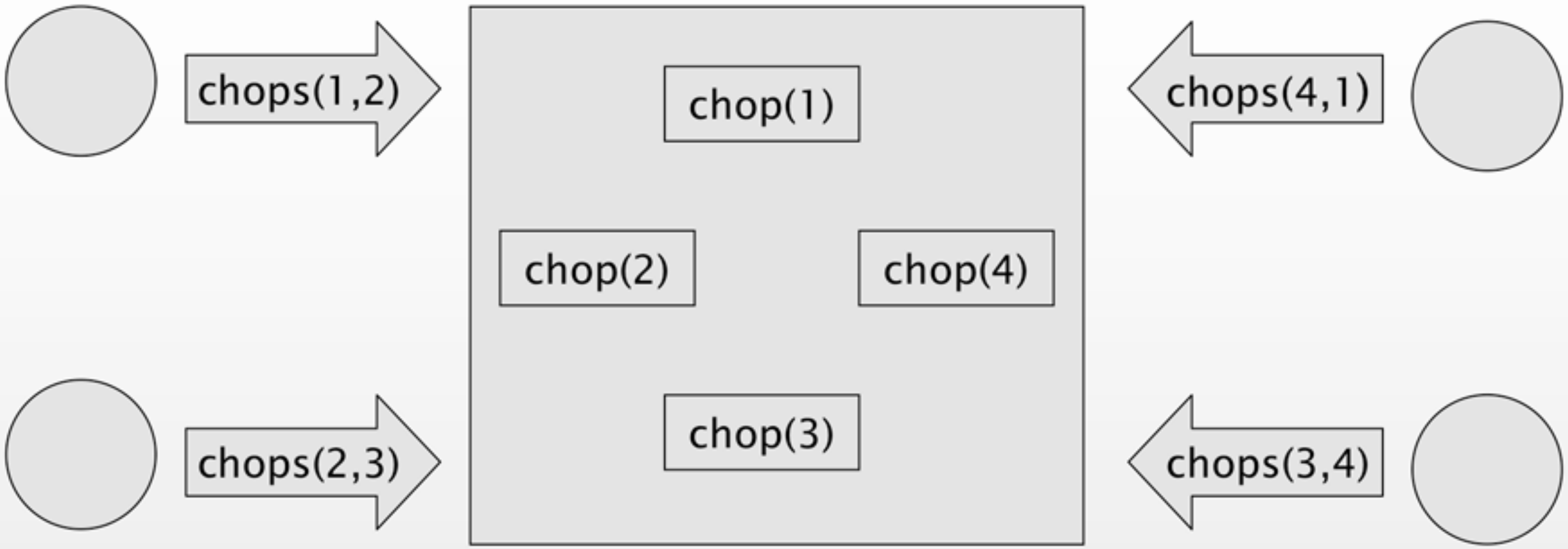
Coordination Artifacts

- A notion of coordination abstraction for MAS
  - inspired by *mediating artifacts* of Activity Theory
  - ➢ artifacts constructed and used by humans to coordinate one another (semaphores, maps, blackboards, signs,...)
  - agents exploit the services of coordination artifacts
- Devised for engineering purposes, featuring:
  - Usage interface & Operating instructions for the agents
  - Inspectability/Adaptability of behaviour
- A coordination artifact is NOT an agent!
  - it does not "achieve goals in autonomy", it is not proactive
  - it calls for a different model, design, implementation
- In TuCSoN, coordination artifacts are realised through ReSpecT tuple centres

ReSpecT Tuple Centres

- Without a specific programming, they are Linda tuple spaces
  - with 1st order logic terms as tuples
  - and logic unification as matching criterion
- They can be programmed with ReSpecT
  - *R*eaction *Spec*ification *T*uples [Omicini & Denti 2001]
  - Defines how to reactively transform the set of tuples as
  - ➢ a new “input event” is received (listening)
  - ➢ a new “output event” is produced (speaking)
- Paradigm
  - transformations through atomic triggered reactions
  - ➢ each of which may trigger new ones

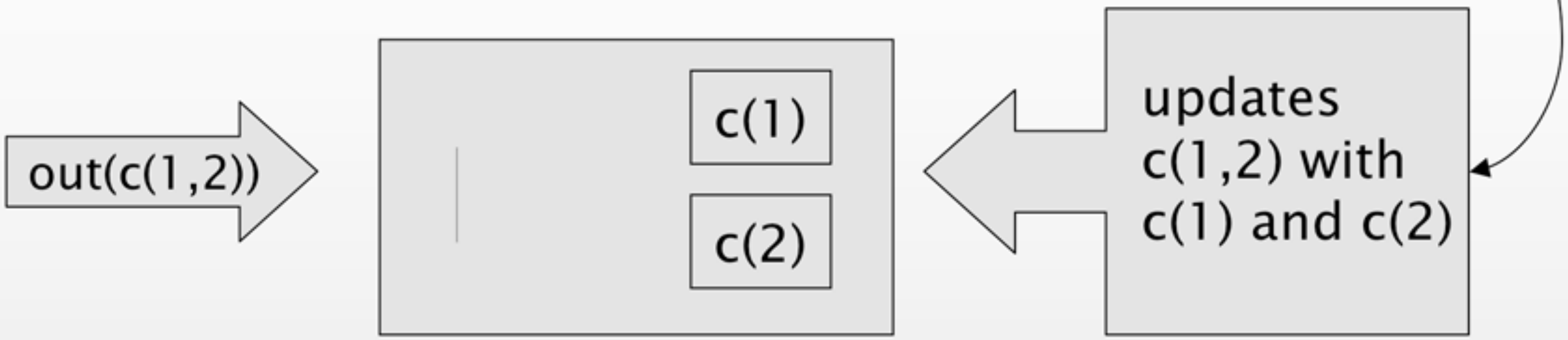
Dining Philosophers



- The problem:
  - Each agent needs to access two locks in an atomic way
  - Locks are shared with another agent
  - ➢ Is a non-trivial example of coordination policy
  - ➢ causes deadlock in Linda (accessing tuples separately)
- The solution using ReSpecT [SAC98]
  - agents put and remove couples of locks
  - internally, couples are divided into single locks

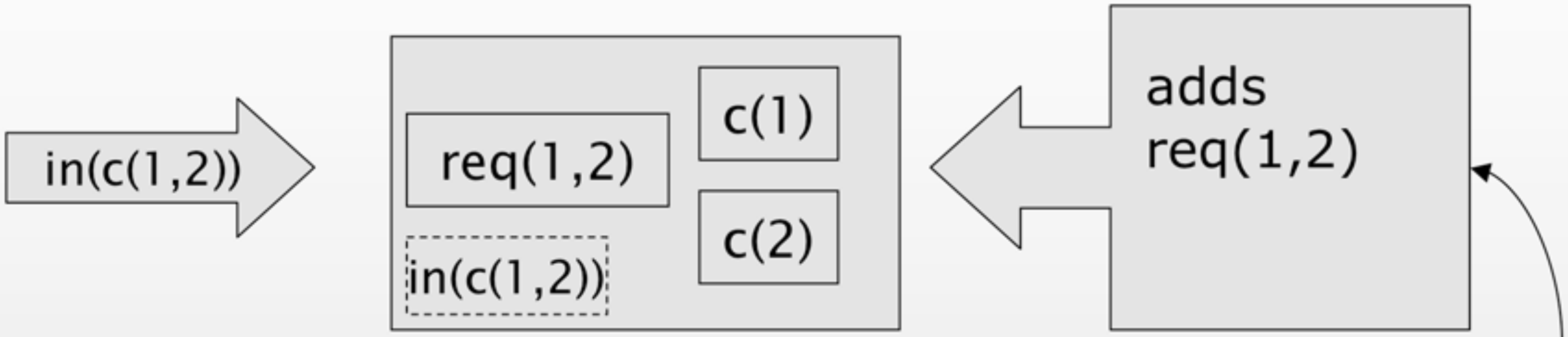
ReSpecT Specification Tuples

reaction(out(c(X1,X2)), (in_r(c(X1,X2)), out_r(c(X1)), out_r(c(X2))))



ReSpecT Specification Tuples

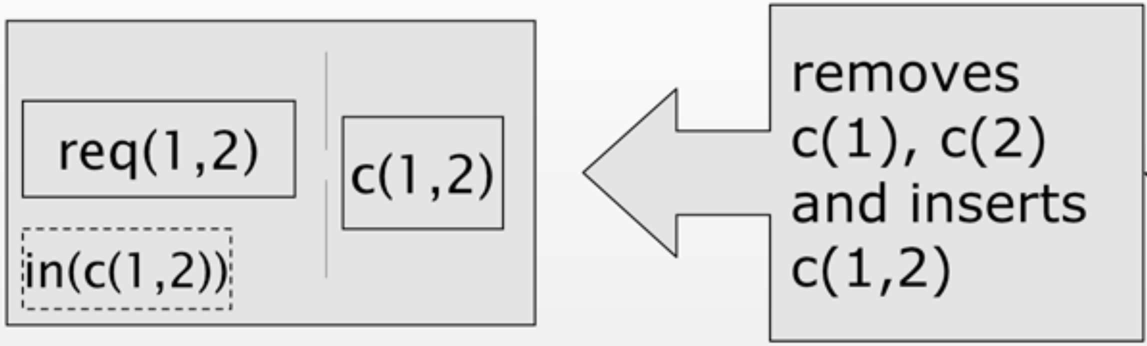
```
reaction( out(c(X1,X2)), ( in_r(c(X1,X2)), out_r(c(X1)), out_r(c(X2)) ))
```




```
reaction(  in(c(X,Y)), ( pre, out_r(req(X,Y)) ))
reaction(  out_r(req(X,Y)), (in_r(c(X)),in_r(c(Y)), out_r(c(X,Y)) ))
```

ReSpecT Specification Tuples

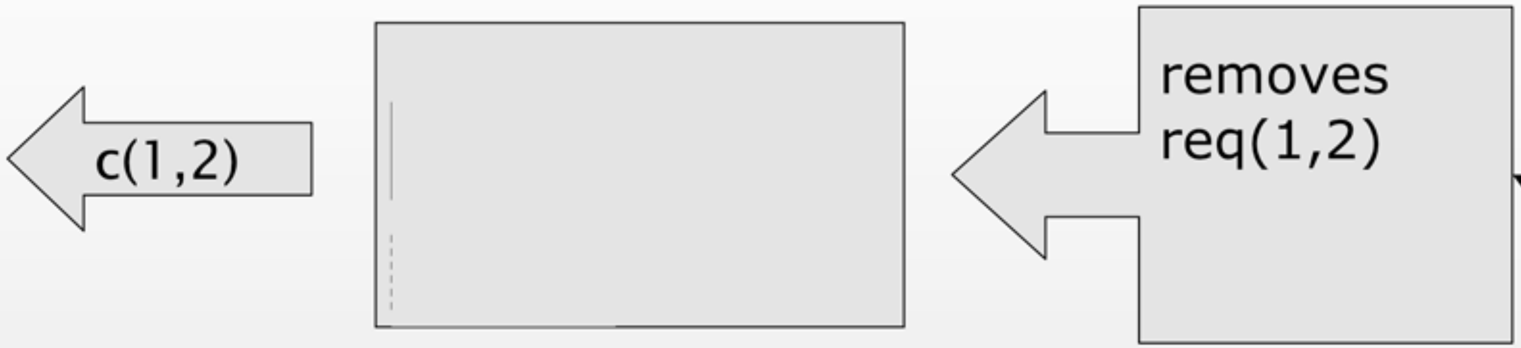
```
reaction( out(c(X1,X2)), ( in_r(c(X1,X2)), out_r(c(X1)), out_r(c(X2)) ))
```




```
reaction(  in(c(X,Y)), ( pre, out_r(req(X,Y)) ))
reaction(  out_r(req(X,Y)), (in_r(c(X)),in_r(c(Y)), out_r(c(X,Y)) ))
```

ReSpecT Specification Tuples

```
reaction( out(c(X1,X2)), ( in_r(c(X1,X2)), out_r(c(X1)), out_r(c(X2)) ))
```




```
reaction(  in(c(X,Y)), ( pre, out_r(req(X,Y)) ))
reaction(  out_r(req(X,Y)), (in_r(c(X)),in_r(c(Y)), out_r(c(X,Y)) ))
```

```
reaction(  in(c(X,Y)), ( post, in_r(req(X,Y)) ))
```

The complete code

```
reaction( out(chops(C1,C2)),
          (in_r(chops(C1,C2)), out_r(chop(C1)), out_r(chop(C2)))).

reaction( in(chops(C1,C2)),
          (pre, out_r(required(C1,C2)))).

reaction( out_r(required(C1,C2)),
          (in_r(chop(C1)), in_r(chop(C2)), out_r(chops(C1,C2)))).

reaction( in(chops(C1,C2)),
          (post, in_r(required(C1,C2)))).

reaction( out_r(chop(C1)),
          (rd_r(required(C1,C), in_r(chop(C1)),
           in_r(chop(C)), out_r(chops(C1,C2)))).

reaction( out_r(chop(C2)),
          (rd_r(required(C,C2), in_r(chop(C)),
           in_r(chop(C2)), out_r(chops(C,C2)))).
```

ReSpecT Syntax

$\sigma ::= \{\texttt{reaction(}p\texttt{(t),(}body\texttt{)).}\}$	Specification
$p ::= cp \mid rp$	ReSpecT primitives
$cp ::= \texttt{out} \mid \texttt{in} \mid \texttt{rd}$	Communication primitives
$rp ::= \texttt{in\_r} \mid \texttt{rd\_r} \mid \texttt{out\_r} \mid \texttt{no\_r}$	Reaction primitives
$body ::= [goal\{\texttt{,}goal\}]$	Specification body
$ph ::= \texttt{pre} \mid \texttt{post}$	Direction predicates
$goal ::= ph \mid rp\texttt{(t)}$	Goals

The ReSpecT Language

- Semantics
  - expresses transformations of tuple sets
  - globally triggered reacting to communication events
  - made of recursive triggering of atomic internal reactions
  - ➢ Turing-complete formalism [Denti, Natali, Omicini 1998]
- Use
  - to make tuple centres automate specific coordination tasks
  - ➢ possibly an assembler for higher-level languages
  - ReSpecT tuple centres as VM for coordination media
- Domains
  - workflow activities
  - protocols enforcement
  - data-oriented forms of cooperation

Timed Coordination

- The notion of time arises in coordination in the context of open and complex systems
- Need for
  - an infrastructure soliciting agent interaction
  - an infrastructure avoiding denial of service due to iper-active agents
  - an agent soliciting infrastructure interaction
- In general
  - we need to specify and enact time-dependent coordination laws (timeouts, delays,...)

Related Approaches

- Technologies
  - JavaSpaces:
    - tuples with a lease-time
    - predicate primitives with a timeout (read, take,..)
- Models
  - Timed Linda [de Boer+Gabbrielli+Meo,1996]
- Formal foundation
  - Interaction
    - Process algebras for timed systems [1995]
  - Coordination
    - JavaSpaces formal model [Zavattaro et.al 2000]
    - Expressiveness of timed coordination languages [Jacquet et.al's 2004]

Extending ReSpecT with Time

Syntax, in prolog-like predicates:

- - for outputs, + for inputs, @ for ground inputs, ? for I/O

Three new primitives

- currentTime(?Tc)
  - Binds variable Tc with the current tuple centre time
  - a time-increasing integer value (millisecs.)
- newTrap(-ID, @Te, +Td)
  - creates a new trap source, with identifier ID
  - which will fire a trap event after Te time units
  - with tuple Td as content of the trap event
- kill_trap(@ID)
  - deallocate the trap source with identifier ID

Trap events listening

- When the Te time expires, the trap event is generated which can be listened by a reaction specification tuple of the kind...
  - reaction(trap(Tuple), Body)
- .. where Tuple is the trap event content tuple

Example 1: Timed In

- A basic extension to the Linda coordination model provides predicate queries (in and rd) with a timeout
  - allow an agent to request information from the infrastructure to be received within a timeout
- Timed In: in(timed(@Time,?Tuple,-Res))
  - ask for removing a tuple matching Tuple
  - within Time units
  - Res will contain the result of removing (yes/no)

Specification (1/4)

```
1  reaction( in(timed(Time,Tuple,Res)), (
     pre, in_r(Tuple),
     out_r(timed(Time,Tuple,yes)))).

2  reaction( in(timed(Time,Tuple,Res)), (
     pre,no_r(Tuple),
     new_trap(ID,Time,expired_in(Time,Tuple)),
     out_r(trap_info(ID,Time,Tuple)) )).

3  reaction( trap(expired_in(Time,Tuple)),(
     in_r(trap_info(ID,Time,Tuple)),
     out_r(timed(Time,Tuple,no)) )).

4  reaction( out(Tuple),(
     in_r(trap_info(ID,Time,Tuple)),
     kill_trap(ID),
     out_r(timed(Time,Tuple,yes)) )).
```

1. As the in(timed(..)) is listened, if Tuple occurs
   - remove the tuple and reify the result timed(Time,Tuple,yes)

Specification (2/4)

```
1  reaction( in(timed(Time,Tuple,Res)), (
     pre, in_r(Tuple),
     out_r(timed(Time,Tuple,yes)))).

2  reaction( in(timed(Time,Tuple,Res)), (
     pre,no_r(Tuple),
     new_trap(ID,Time,expired_in(Time,Tuple)),
     out_r(trap_info(ID,Time,Tuple)) )).

3  reaction( trap(expired_in(Time,Tuple)),(
     in_r(trap_info(ID,Time,Tuple)),
     out_r(timed(Time,Tuple,no)) )).

4  reaction( out(Tuple),(
     in_r(trap_info(ID,Time,Tuple)),
     kill_trap(ID),
     out_r(timed(Time,Tuple,yes)) )).
```

2. As the in(timed(..)) is listened, if Tuple does not occur
   - generate the trap source (expired_in)
   - reify info on the trap (trap_info)

Specification (3/4)

```
1  reaction( in(timed(Time,Tuple,Res)), (
     pre, in_r(Tuple),
     out_r(timed(Time,Tuple,yes)))).

2  reaction( in(timed(Time,Tuple,Res)), (
     pre,no_r(Tuple),
     new_trap(ID,Time,expired_in(Time,Tuple)),
     out_r(trap_info(ID,Time,Tuple)) )).

3  reaction( trap(expired_in(Time,Tuple)),(
     in_r(trap_info(ID,Time,Tuple)),
     out_r(timed(Time,Tuple,no)) )).

4  reaction( out(Tuple),(
     in_r(trap_info(ID,Time,Tuple)),
     kill_trap(ID),
     out_r(timed(Time,Tuple,yes)) )).
```

3. If the trap event is generated
   - remove the reified info on the trap source
   - reify the result timed(Time,Tuple,no)

Specification (4/4)

```
1  reaction( in(timed(Time,Tuple,Res)), (
     pre, in_r(Tuple),
     out_r(timed(Time,Tuple,yes)))).

2  reaction( in(timed(Time,Tuple,Res)), (
     pre,no_r(Tuple),
     new_trap(ID,Time,expired_in(Time,Tuple)),
     out_r(trap_info(ID,Time,Tuple)) )).

3  reaction( trap(expired_in(Time,Tuple)),(
     in_r(trap_info(ID,Time,Tuple)),
     out_r(timed(Time,Tuple,no)) )).

4  reaction( out(Tuple),(
     in_r(trap_info(ID,Time,Tuple)),
     kill_trap(ID),
     out_r(timed(Time,Tuple,yes)) )).
```

4. If a matching tuple is inserted in the space
    - finds a pending trap source that matches
    - kill its trap and reify a positive result

Tuples with Lease

```
1  reaction( out(leased(Time,Tuple)), (
     new_trap(ID,Time,lease_expired(Time,Tuple)),
     in_r(leased(Time,Tuple)),
     out_r(outl(ID,Time,Tuple)) )).

2  reaction( rd(Tuple),( pre,
     rd_r(outl(ID,_,Tuple)),
     out_r(Tuple) )).

3  reaction( rd(Tuple),(post,
     rd_r(outl(ID,_,Tuple)),
     in_r(Tuple) )).

4  reaction( in(Tuple),( pre,
     in_r(outl(ID,_,Tuple)),
     out_r(Tuple),
     kill_trap(ID) )).

5  reaction( trap(lease_expired(Time,Tuple)), (
     in_r(outl(ID,Time,Tuple)))).
```

As the tuple is inserted generates the trap source

Handle successful reads of the tuple

Handle unsuccessful reads of the tuple

Handle removals of the tuple

Lease expiring

Timed Dining Philosophers

- Is an extension of the dining philosophers case
  - exemplifies the need for adding time constraints to an exiting, complex coordination scenario
- A tuple in the tuple centre stores the maximum amount of time which an agent can need for using the resource (eat)
  - max_eating_time(Time)
  - if this expires the locks are automatically released (chopsticks are re-inserted)
  - late releases are to be consumed

Specification

```
reaction(in(chops(C1,C2)),( pre,
  rd_r(max_eating_time(Tmax)),
  new_trap(ID,Tmax, expired(C1,C2)),
  current_agent(AgentId),
  out_r(chops_pending_trap(ID,AgentId,C1,C2)))).

reaction(out(chops(C1,C2)),(
  in_r(chops_pending_trap(ID,C1,C2)),
  kill_trap(ID))).

reaction(trap(expired(C1,C2)),(
  no_r(chop(C1)), no_r(chop(C2)),
  current_agent(AgentId),
  in_r(chops_pending_trap(ID,AgentId,C1,C2)),
  out_r(invalid_chops(AgentId,C1,C2)),
  out_r(chop(C1)), out_r(chop(C2)))).

reaction(out(chops(C1,C2)), (
  current_agent(AgentId),
  in_r(invalid_chops(AgentId,C1,C2)),
  in_r(chops(C1,C2)))).
```

When chopsticks are consumed, a trap generator is

As the chopsticks are released, the generator is killed

As the trap event is listened, chopsticks are re-inserted!

Late re-insertions of chopsticks are ignored!

- Rules to be modulary added to the untimed spec.

Timed Contract Net

- CNP
  - A master announces a task to be executed
  - workers provide their bids
  - one of them is selected which executes the task
- Timed extension to guarantee liveness. We add timeouts for
  - the bidding stage
  - the master to communicate the awarded worker
  - the awarded worker to confirm its bid
  - the awarded worker to execute the task

Specification

```
% When an announcement is made, a trap generator is
% installed for generating a timeout for bidding time
1  reaction(out(announcement(task(Id,Info,MaxTime))),(
     out_r(task_todo(Id,Info,MaxTime)),
     out_r(cnp_state(collecting_bids(Id))),
     rd_r(bidding_time(Time)),
     new_trap(_,Time,bidding_expired(Id)))).

% When the bidding time has expired, the master can
% collect the bids for choosing the winner. A trap
% generator is installed for defining the maximum
% awarding time
2  reaction(trap(bidding_expired(TaskId)),(
      in_r(announcement(_)),
      in_r(cnp_state(collecting_bids(TaskId))),
      out_r(collected_bids(TaskId,[])),
      out_r(cnp_state(awarding(TaskId))),
      rd_r(awarding_time(Time)),
      new_trap(_,Time,awarding_expired(TaskId)))).
3  reaction(out_r(collected_bids(TaskId,L)),(
      in_r(bid(TaskId,AgentId,Bid)),
      out_r(bid_evaluated(TaskId,AgentId,Bid)),
      in_r(collected_bids(TaskId,L)),
      out_r(collected_bids(TaskId,
               [bid(AgentId,Bid)|L])) )).

% When the awarding time has expired, the bidders are
% informed of the results. If no winner has been
% selected the protocol enters in an error state,
% otherwise the protocol enters in the confirming
% stage, setting up a maximum time for it
4  reaction(trap(awarding_expired(TaskId)),(
      in_r(cnp_state(awarding(TaskId))),
      out_r(check_awarded(TaskId)))).
5  reaction(out_r(check_awarded(TaskId)),(
      in_r(check_awarded(TaskId)),
      rd_r(awarded_bid(TaskId,AgentId)),
      in_r(bid_evaluated(TaskId,AgentId,Bid)),
      out_r(result(TaskId,AgentId,awarded)),
      out_r(cnp_state(confirming_bid(TaskId,AgentId))),
      rd_r(confirming_time(Time)),
      new_trap(ID,Time,confirm_expired(TaskId)),
      out_r(confirm_timer(TaskId,ID)),
      out_r(refuse_others(TaskId)))).
6  reaction(out_r(check_awarded(TaskId)),(
      in_r(check_awarded(TaskId)),
      rd_r(awarded_bid(TaskId,AgentId)),
      no_r(bid_evaluated(AgentId,Bid)),
      out_r(cnp_state(aborted(TaskId,wrong_awarded))))).
7  reaction(out_r(check_awarded(TaskId)),(
      in_r(check_awarded(TaskId)),
      no_r(awarded_bid(TaskId,AgentId)),
      out_r(cnp_state(aborted(TaskId,award_expired))))).
8  reaction(out_r(refuse_others(TaskId)),(
      in_r(bid_evaluated(TaskId,AgentId,Bid)),
      out_r(result(TaskId,AgentId,'not-awarded')),
      out_r(refuse_others(TaskId)))).
9  reaction(out_r(refuse_others(TaskId)),(
      in_r(refuse_others(TaskId)) )).

% At the arrival of the confirm from the awarded
% bidder, a timeout trap is setup for checking the
% execution time of the task
10  reaction(out(confirm_bid(TaskId,AgentId)),(
      in_r(confirm_bid(TaskId,AgentId)),
      in_r(cnp_state(confirming_bid(TaskId,AgentId))),
      current_time(StartTime),
      out_r(cnp_state(executing_task(TaskId,StartTime))),
      in_r(confirm_timer(TaskId,IdT)),
      kill_trap(IdT),
      rd_r(task_todo(TaskId,_,MaxTime)),
      new_trap(IdT2, MaxTime, execution_expired),
      out_r(execution_timer(TaskId,IdT2)))).

% The occurrence of the confirm expired trap means
% that the confirm from the awarded bidder has not
% arrived on time, causing the protocol to be aborted
11  reaction(trap(confirm_expired(TaskId)),(
      in_r(cnp_state(confirming_bid(TaskId,AgentId))),
      in_r(confirm_timer(TaskId,_)),
      rd_r(awarded_bid(TaskId,AgentId)),
      out_r(cnp_state(aborted(TaskId,
               confirm_expired(AgentId)))))).

% The occurrence of the execution expired trap means
% that the awarded bidder has not completed the
% task on time, causing the protocol to be aborted
12  reaction(trap(execution_expired(TaskId)),(
      in_r(cnp_state(executing_task(TaskId,StartTime))),
      in_r(execution_timer(TaskId,_)),
      rd_r(awarded_bid(TaskId,AgentId)),
      current_time(Now),
      Duration is Now - StartTime,
      out_r(cnp_state(aborted(TaskId,
               execution_expired(AgentId,Duration)))))).

% The awarded bidder provided task result on time
% terminating correctly the protocol
13  reaction(out(task_result(TaskId,AgentId,Result)),(
      in_r(task_result(TaskId,AgentId,Result)),
      in_r(awarded_bid(TaskId,AgentId)),
      in_r(execution_timer(TaskId,Id)),
      kill_trap(Id),
      in_r(cnp_state(executing_task(TaskId,StartTime))),
      in_r(task_todo(TaskId,Info,MaxTime)),
      current_time(Now),
      Duration is Now - StartTime,
      out_r(task_done(TaskId,Result,Duration)) )).
```

It's a reasonable extension to the un-timed specification

Conclusion

- From untimed to timed VM for coordination laws in MAS
  - prelimary proposal, with good support to scalability of coordination law complexity
  - already implemented in TuCSoN 1.4
  - Visit and try: **tucson.sourceforge**.net
- Future works
  - More practice and experience with time
  - Deepening internal priority and synchrony issues
  - Full formal model of Timed ReSpecT
  - General redesign of the ReSpecT model, with better integration of time, observation & meta-level

Timed Coordination Artifacts with ReSpecT

Mirko Viroli, Alessandro Ricci
DEIS, Sede di Cesena
Alma Mater Studiorum, Università di Bologna
{mviroli + aricci}@deis.unibo.it
properly filtered by Andrea Omicini