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
  – Reaction Specification Tuples [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)) ))

out(c(1,2)) c(1) c(2) c(1,2) with c(1) and c(2)

updates c(1,2) with c(1) and c(2)
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)) ))

removes c(1), c(2) and inserts c(1,2)
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, C1), 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

\[ \sigma ::= \{ \text{reaction}(p(t),(body)). \} \quad \text{Specification} \]
\[ p ::= cp \mid rp \quad \text{ReSpecT primitives} \]
\[ cp ::= \text{out} \mid \text{in} \mid \text{rd} \quad \text{Communication primitives} \]
\[ rp ::= \text{in}_r \mid \text{rd}_r \mid \text{out}_r \mid \text{no}_r \quad \text{Reaction primitives} \]
\[ body ::= [\text{goal}{,\text{goal}}] \quad \text{Specification body} \]
\[ ph ::= \text{pre} \mid \text{post} \quad \text{Direction predicates} \]
\[ goal ::= ph \mid rp(t) \quad \text{Goals} \]
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)
1. As the \texttt{in(timed(..))} is listened, if \texttt{Tuple} occurs
   - remove the tuple and reify the result \texttt{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)
3. If the trap event is generated
   • remove the reified info on the trap source
   • reify the result timed(Time,Tuple,no)
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)))).
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 created.
- 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

It’s a reasonable extension to the untimed specification

WOA 2004, Torino
Mirko Viroli, Alessandro Ricci
Conclusion

• From untimed to timed VM for coordination laws in MAS
  - preliminary 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