Tuple-based Coordination of Situated Systems

Distributed Systems

Sistemi Distribuiti

Andrea Omicini

andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015

Outline

Outline

Situatedness in the Spatio-Temporal Fabric I

What is Situatedness?

- Situatedness is in short the property of systems of being *immersed* in their environment
- That is, of being capable to *perceive* and *produce* environment change, by suitably dealing with environment events
- Mobile, adaptive, and pervasive computing systems have emphasised the key role of situatedness for nowadays computational systems [ZCF+11]

Situatedness in the Spatio-Temporal Fabric II

Situatedness & context-awareness

- Computational systems are more and more affected by their physical nature
- This is not due to a lack of abstraction: indeed, we are suitably layering systems – including hardware and software layers –, where separation between the different layers is clear and well defined
- Instead, this is mostly due to the increasingly complex requirements for our computational systems, which mandate for an ever-increasing *context-awareness* [BDR07]
- Defining the notion of *context* is a complex matter [Omi02], with its boundaries typically blurred with the notion of *environment*, in particular in the field of multi-agent systems [WOO07]

Situatedness in the Spatio-Temporal Fabric III

Context-awareness: space & time

- Mobile computing scenarios have made clear that *situatedness* of computational systems nowadays requires at least *awareness of the spatio-temporal fabric*
- That is, any non-trivial system needs to know *where* it is working, and *when*, in order to effectively perform its function
- In its most general acceptation, then, any environment for a computational systems is first of all made of space and time

Space & time vs. coordination

- Why, and to which extent is this a *coordination issue*?
- Why, and to which extent is this a *tuple-based coordination issue*?

Outline

Dining Philosophers in ReSpecT: Starvation?

What is the problem?

- The problem is *time*: no one keeps track of time here, and starvation is a matter of time
- How can we handle time here? Is synchronisation not enough for the purpose?
- Of course not: to avoid problems like starvation, we need the ability of defining *time-dependent* coordination policies

What is the solution?

- In order to define time-dependent coordination policies, a time-aware coordination medium is needed

Time-aware Coordination Media I

Requirements for a time-aware coordination media

A time-aware coordination medium should satisfy the following requirements [ORV07]:

1. Time has to be an integral part of the ontology of a coordination medium
2. (Physical) time has to be explicitly embedded into the coordination medium working cycle
3. A coordination medium should allow coordination policies to talk about time
4. A coordination medium should be able to capture time events, and to react appropriately
5. A coordination medium should allow coordination policies to be changed over time

Time-aware Coordination Media II

Physical time in the medium ontology & working cycle *(reqs. 1, 2)*

1. Time has to be an integral part of the ontology of a coordination medium
2. (Physical) time has to be explicitly embedded into the coordination medium working cycle

- The ReSpecT admissible event model is extended to include *time*. For instance, on the case of $\langle OpEvent \rangle$:

$$\begin{aligned} \langle OpStartCause \rangle &::= \langle CoordOp \rangle , \langle AgentId \rangle , \langle TCId \rangle , \langle Time \rangle \\ \langle OpEventCause \rangle &::= \langle OpStartCause \rangle \mid \\ &\quad \langle LinkOp \rangle , \langle TCId \rangle , \langle TCId \rangle , \langle Time \rangle \end{aligned}$$

- Every ReSpecT VM executes on a sequential machine making an expression of physical time available

→ At every transition of the ReSpecT VM, physical time is always available to any ReSpecT computation

Time-aware Coordination Media III

Coordination policies: talking about time *(req. 3)*

3. A coordination medium should allow coordination policies to talk about time

- ReSpecT observation predicates are extended with time:

$$\begin{aligned} \langle \textit{ObservationPredicate} \rangle &::= \langle \textit{EventView} \rangle \_ \langle \textit{EventInformation} \rangle \\ \langle \textit{EventView} \rangle &::= \texttt{current} \mid \texttt{event} \mid \texttt{start} \\ \langle \textit{EventInformation} \rangle &::= \ldots \mid \texttt{time} \mid \ldots \end{aligned}$$

- Two ReSpecT guard predicates are introduced:

$$\langle \textit{GuardPredicate} \rangle ::= \ldots \mid \texttt{before(} \langle \textit{Time} \rangle \texttt{)} \mid \texttt{after(} \langle \textit{Time} \rangle \texttt{)} \mid \ldots$$

- along with the obvious alias

$$\texttt{between(} \langle \textit{Time} \rangle \texttt{,} \langle \textit{Time} \rangle \texttt{)}$$

Time-aware Coordination Media IV

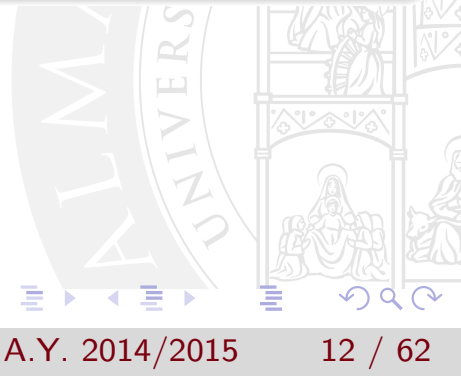
Coordination policies: talking about time *(req. 3 – contd.)*

In the overall, ReSpecT is extended with time

- by introducing some temporal predicates and guards to get information about both tuple-centre and event time

observation `current_time(?Time)`, `start_time(?Time)`, `event_time(?Time)`

guard `before(@Time)`, `after(@Time)`, `between(@MinTime,@MaxTime)`



Time-aware Coordination Media V

Capturing time events *(req. 4)*

4. A coordination medium should be able to capture time events, and to react appropriately

- The ReSpecT *admissible (tuple centre) event* model is extended to include time events:

$$
\begin{array}{rcl}
\langle \mathit{TCEvent} \rangle & ::= & \langle \mathit{OpEvent} \rangle \mid \langle \mathit{TEvent} \rangle \mid \ldots \\
\langle \mathit{TEvent} \rangle & ::= & \langle \mathit{TStartCause} \rangle \, , \langle \mathit{TEventCause} \rangle \, , \langle \mathit{TResult} \rangle \, , \langle \mathit{Time} \rangle \\
\langle \mathit{TStartCause} \rangle & ::= & \langle \mathit{TOp} \rangle \, \texttt{,time,} \, \langle \mathit{TCId} \rangle \\
\langle \mathit{TEventCause} \rangle & ::= & \langle \mathit{TStartCause} \rangle \\
\langle \mathit{TOp} \rangle & ::= & \texttt{time(} \, \langle \mathit{Time} \rangle \, \texttt{)} \\
\langle \mathit{TResult} \rangle & ::= & \langle \mathit{TOp} \rangle \, , \ldots
\end{array}
$$

- Correspondingly, the ReSpecT *event descriptor* is extended, too:

$$
\langle \mathit{Event} \rangle \quad ::= \quad \langle \mathit{Predicate} \rangle \, \texttt{(} \, \langle \mathit{Tuple} \rangle \, \texttt{)} \mid \texttt{time(} \, \langle \mathit{Time} \rangle \, \texttt{)} \mid \ldots
$$

making it possible to specify reactions to *time events*:

```
reaction(time(@Time), Guard, Body).
```

Time-aware Coordination Media VI

Changing coordination policies over time *(req. 5)*

5. A coordination medium should allow coordination policies to be changed over time

- It is enough to exploit malleabilty of ReSpecT tuple centres
- exploiting the same $\langle ForgePredicate \rangle$s that can be generally used for dynamically forge tuple centre behaviour at run time
- such as `in_s`, `out_s`, ...

Timed Dining Philosophers

- An example of time-dependent coordination
- `table` tuple centre stores the maximum amount of time for any process (philosopher) to use the resource (to eat using chops)
  - in terms of a tuple `max_eating_time(@Time)`
  - if this time expires the locks are automatically released—chopsticks are re-inserted by the table tuple centre
  - late releases (by processes through `seat` tuple centres) are to be ignored—linkability used to make `seat` tuple centres consistent
- With a very simple extension using timed reactions, Distributed Timed Dining Philosophers are done
  - see [ORV05]

Timed Dining Philosophers: Philosopher

```
philosopher(I,J) :-
    think,                          % thinking
    table ? in(chops(I,J)),         % waiting to eat
    eat,                            % eating
    table ? out(chops(I,J)),        % waiting to think
!,  philosopher(I,J).
```

Timed Dining Philosophers: Philosopher

```
philosopher(I,J) :-
    think,                          % thinking
    table ? in(chops(I,J)),         % waiting to eat
    eat,                            % eating
    table ? out(chops(I,J)),        % waiting to think
!,  philosopher(I,J).
```

With respect to Dining Philosopher's protocol...

...this is left unchanged

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)), out(chops(C1,C2)) )).
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)), out(chops(C1,C2)) )).
reaction( out(chop(C)), internal, (                          % (5)
    rd(required(C,C2)), in(chop(C)), in(chop(C2)), out(chops(C,C2)) )
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)), out(chops(C1,C2)) )).
reaction( out(chop(C)), internal, (                          % (5)
    rd(required(C,C2)), in(chop(C)), in(chop(C2)), out(chops(C,C2)) )
reaction( out(chop(C)), internal, (                          % (5')
    rd(required(C1,C)), in(chop(C1)), in(chop(C)), out(chops(C1,C)) )
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (     % (1)
    in(chops(C1,C2)) )).
reaction( out(chops(C1,C2)), (operation, completion), (     % (1')
    out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (      % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (      % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                 % (4)
    in(chop(C1)), in(chop(C2)), out(chops(C1,C2)) )).
reaction( out(chop(C)), internal, (                         % (5)
    rd(required(C,C2)), in(chop(C)), in(chop(C2)), out(chops(C,C2)) )).
reaction( out(chop(C)), internal, (                         % (5')
    rd(required(C1,C)), in(chop(C1)), in(chop(C)), out(chops(C1,C)) )).
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)) )).
reaction( out(chops(C1,C2)), (operation, completion), (      % (1')
    in(used(C1,C2,_)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)), out(chops(C1,C2)) )).
reaction( out(chop(C)), internal, (                          % (5)
    rd(required(C,C2)), in(chop(C)), in(chop(C2)), out(chops(C,C2)) )).
reaction( out(chop(C)), internal, (                          % (5')
    rd(required(C1,C)), in(chop(C1)), in(chop(C)), out(chops(C1,C)) )).
```

Timed Dining Philosophers: table ReSpecT Code

```
reaction( out(chops(C1,C2)), (operation, completion), (      % (1)
    in(chops(C1,C2)) )).
reaction( out(chops(C1,C2)), (operation, completion), (      % (1')
    in(used(C1,C2,_)), out(chop(C1)), out(chop(C2)) )).
reaction( in(chops(C1,C2)), (operation, invocation), (       % (2)
    out(required(C1,C2)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (3)
    in(required(C1,C2)) )).
reaction( out(required(C1,C2)), internal, (                  % (4)
    in(chop(C1)), in(chop(C2)), out(chops(C1,C2)) )).
reaction( out(chop(C)), internal, (                          % (5)
    rd(required(C,C2)), in(chop(C)), in(chop(C2)), out(chops(C,C2)) )).
reaction( out(chop(C)), internal, (                          % (5')
    rd(required(C1,C)), in(chop(C1)), in(chop(C)), out(chops(C1,C)) )).
reaction( in(chops(C1,C2)), (operation, completion), (       % (6)
    current_time(T), rd(max eating time(Max)), T1 is T + Max,
    out(used(C1,C2,T)),
    out_s(time(T1),(in(used(C1,C2,T)), out(chop(C1)), out(chop(C2))))) ).
```

Timed Dining Philosophers in ReSpecT: Results

Results

protocol	no deadlock
protocol	fairness
protocol	trivial philosopher's interaction protocol
tuple centre	shared resources handled properly
tuple centre	no starvation

Outline

What About Coordination & Space?

The issue of space awareness

- The availability of a plethora of mobile devices, in particular, is pushing forward the needs for space-awareness of computations and systems
- Awareness of the *spatial context* is often essential to establish which tasks to perform, which goals to achieve, and how
- More generally, spatial issues are fundamental in many sorts of complex software systems, including intelligent, multi-agent, adaptive, and self-organising ones [Bea10]
- In most of the application scenarios where situatedness plays an essential role, computation and coordination are required to be *space aware*

Situatedness & Awareness I

- Spatial coordination requires
  - spatial situatedness
  - spatial awareness

  of the coordination media
- This translates in a number of *technical requirements*

Situatedness & Awareness II

Situatedness

A *space-aware coordination abstraction* should at any time be associated to an absolute positioning, both physical and virtual

In fact,

- *software abstractions* may move along a *virtual space* – typically, the network – which is usually *discrete*
- whereas *physical devices* move through a *physical space*, which is mostly *continuous*

However, software abstractions may also be hosted by mobile physical devices, thus *share* their motion.

Situatedness & Awareness III

Awareness

The position of the coordination medium should be available to the *coordination laws* it contains in order to make them capable of *reasoning about space*—so, to implement space-aware coordination laws.

Also, space has to be embedded into the working cycle of the coordination medium:

- a spatial event should be *generated* within a coordination medium, conceptually corresponding to *changes in space*
- then, such events should be *captured* by the coordination medium, and used to *activate* space-aware coordination laws

Space-aware Coordination Medium: Requirements I

Situatedness: Requirements

1. Space should intrinsically belong to the ontology of the coordination medium, in terms of *position* and *motion*—any sort of motion information, such as speed, acceleration, ...
2. Both *virtual* and *physical* space acceptations should be supported
3. A notion of *locality* should be made available

Awareness: Requirements

4. A coordination medium should allow coordination policies to talk about space
5. A coordination medium should be able to capture spatial events, and to react appropriately

ReSpecT Tuple Centres as Space-aware Media I

Space in medium ontology & working cycle *(reqs. 1, 2)*

1. Space has to be an integral part of the ontology of a coordination medium
2. Both physical & virtual position & motion have to be explicitly embedded into the coordination medium working cycle

- The ReSpecT admissible event model is extended to include *space*—physical & virtual. For instance, on the case of $\langle OpEvent \rangle$:

$$\begin{array}{rcl} \langle OpStartCause \rangle & ::= & \langle CoordOp \rangle , \langle AgentId \rangle , \langle TCId \rangle , \langle Time \rangle , \langle Place \rangle , \langle Node \rangle \\ \langle OpEventCause \rangle & ::= & \langle OpStartCause \rangle \mid \\ & & \langle LinkOp \rangle , \langle TCId \rangle , \langle TCId \rangle , \langle Time \rangle , \langle Place \rangle , \langle Node \rangle \end{array}$$

- Every ReSpecT VM executes on a physical machinery and network node making an expression of physical & virtual positioning available

$\rightarrow$ At every transition of the ReSpecT VM, physical & virtual positioning is always available to any ReSpecT computation

ReSpecT Tuple Centres as Space-aware Media II

Locality in TuCSoN *(req. 3)*

3. A notion of *locality* should be made available

- ReSpecT tuple centres have unique identities and IDs within a TuCSoN node
- any $\langle CoordOp \rangle$ / $\langle LinkOp \rangle$ can refer to a simple tuple centre identifier, using a TuCSoN node as a straightforward *local (coordination) space*
- adding a node reference to a $\langle CoordOp \rangle$ / $\langle LinkOp \rangle$ shifts everything to the *global (coordination) space*
- this however is just a notion of virtual locality

ReSpecT Tuple Centres as Space-aware Media III

Coordination policies: talking about space *(req. 4)*

4. A coordination medium should allow coordination policies to talk about space

- ReSpecT observation predicates are extended with physical & virtual space:

$$\begin{aligned} \langle \textit{ObservationPredicate} \rangle &::= \langle \textit{EventView} \rangle \_ \langle \textit{EventInformation} \rangle \\ \langle \textit{EventView} \rangle &::= \texttt{current} \mid \texttt{event} \mid \texttt{start} \\ \langle \textit{EventInformation} \rangle &::= \ldots \mid \texttt{place} \mid \texttt{node} \mid \ldots \end{aligned}$$

- Three ReSpecT guard predicates are introduced:

$$\langle \textit{GuardPredicate} \rangle ::= \ldots \mid \texttt{at(} \langle \textit{Place} \rangle \texttt{)} \mid \texttt{near(} \langle \textit{Place} \rangle \texttt{,} \langle \textit{Radius} \rangle \texttt{)} \mid \texttt{on(} \langle \textit{Node} \rangle \texttt{)} \mid \ldots$$

ReSpecT Tuple Centres as Space-aware Media IV

Coordination policies: talking about space *(req. 4 – contd.)*

In the overall, ReSpecT is extended with space

- by introducing some spatial predicates and guards to get information about both tuple-centre and event physical & virtual positioning

observation `current_place(?Place)`, `start_place(?Place)`, `event_place(?Place)`, `current_node(?Node)`, `start_node(?Node)`, `event_node(?Node)`

guard `at(@Place)`, `near(@Place,@Radius)`, `on(@Node)`

ReSpecT Tuple Centres as Space-aware Media V

Capturing spatial events *(req. 5)*

6. A coordination medium should be able to capture spatial events, and to react appropriately

- The ReSpecT *admissible (tuple centre) event* model is extended to include spatial events

$$
\begin{array}{rcl}
\langle TCEvent \rangle & ::= & \langle OpEvent \rangle \mid \langle TEvent \rangle \mid \langle SEvent \rangle \mid \ldots \\
\langle SEvent \rangle & ::= & \langle SStartCause \rangle \, , \langle SEventCause \rangle \, , \langle SResult \rangle \, , \\
 & & \langle Time \rangle \, , \langle Place \rangle \, , \langle Node \rangle \\
\langle SStartCause \rangle & ::= & \langle SOp \rangle \, , \texttt{space} , \langle TCId \rangle \\
\langle SEventCause \rangle & ::= & \langle SStartCause \rangle \\
\langle SOp \rangle & ::= & \texttt{from(} \langle Place \rangle \texttt{)} \mid \texttt{to(} \langle Place \rangle \texttt{)} \mid \texttt{node(} \langle Node \rangle \texttt{)} \\
\langle SResult \rangle & ::= & \langle SOp \rangle \, , \ldots
\end{array}
$$

ReSpecT Tuple Centres as Space-aware Media VI

Capturing spatial events *(req. 5 – contd.)*

- Correspondingly, the ReSpecT *event descriptor* is extended, too

$$\langle \textit{Event} \rangle \quad ::= \quad \langle \textit{Predicate} \rangle \texttt{(} \langle \textit{Tuple} \rangle \texttt{)} \mid \texttt{time(} \langle \textit{Time} \rangle \texttt{)} \mid \texttt{from(} \langle \textit{Place} \rangle \texttt{)} \mid \texttt{to(} \langle \textit{Place} \rangle \texttt{)} \mid \texttt{node(} \langle \textit{Node} \rangle \texttt{)} \mid \ldots$$

thus making it possible to specify reactions to the occurrence of *spatial events*

```
reaction(from(?Place), Guard, Body).
reaction(to(?Place), Guard, Body).
reaction(node(?Node), Guard, Body).
```

Spatial Observation Predicates

`current_place(@S,?P)` succeeds if P unifies with the position of the node which the tuple centre belongs to

`event_place(@S,?P)` succeeds if P unifies with the position of the node where the triggering event was originated

`start_place(@S,?P)` succeeds if P unifies with the position of the node where the event chain that led to the triggering event was originated

The node position can be specified as either

`S=ph` its absolute physical position

`S=ip` its IP number

`S=dns` its domain name

`S=map` its geographical location—as typically defined by mapping services like Google Maps

`S=org` its organisational position—that is, a location within an organisation-defined virtual topology

Spatial Guard Predicates

`at(@S,@P)` succeeds when the tuple centre is currently executing at the position P, specified according to S

`near(@S,@P,@R)` succeeds when the tuple centre is currently executing at the position included in the spatial region with centre P and radius R, specified according to S

Spatial Event Descriptors

`from(?S,?P)` matches a spatial event raised when the device hosting the tuple centre starts moving from position P, specified according to S.

`to(?S,?P)` matches a spatial event raised when the device hosting the tuple centre stops moving and reaches position P, specified according to S.

As a result, the following are admissible reaction specification tuples dealing with spatial events:

```
reaction(from(?Space,?Place), Guards, Reactions).
reaction(to(?Space,?Place), Guards, Reactions).
```

Outline

Outline

Situatedness & Coordination I

Situatedness...

- essentially, strict coupling with the environment
- technically, the ability to properly perceive and react to changes in the environment
- one of the most critical issues in distributed systems
  - conceptual clash between pro-activeness in process behaviour and reactivity w.r.t. environment change
- still one of the most critical issues for artificial intelligence & robotics [Suc87]

Situatedness & Coordination II

. . . & coordination

- essentially, situatedness concerns interaction between processes and the environment
- technically, situatedness can be conceived as a coordination problem
  - how to handle and govern interaction between pro-active processes and an ever-changing environment

Governing interaction

- Intra-system interaction via coordination media as rulers of component-component interaction
- Inter-system interaction via. . . ?
  - coordination media as rulers of component-environment interaction

Outline

Situating ReSpecT I

ReSpecT tuple centres for environment engineering

- Distributed systems are immersed into an environment, and should be reactive to events of *any* sort
- Also, coordination media should mediate any activity toward the environment, allowing for a fruitful interaction

$\Rightarrow$ ReSpecT tuple centres should be able to *capture general environment events*, and to generally *mediate process-environment interaction*

Situating ReSpecT II

Situating ReSpecT

- Thus, *situating* the ReSpecT language basically means making ReSpecT capable of *capturing environment events*, and *expressing general MAS-environment interactions* [CO09]

$\Rightarrow$ ReSpecT captures, reacts to, and observes general environment events

$\Rightarrow$ ReSpecT can explicitly interact with the environment

ReSpecT Tuple Centres as Environment-aware Media I

Coordination policies: talking about environment

1. A coordination medium should allow coordination policies to talk about environment

- ReSpecT observation predicates are extended with environment:

$$\begin{aligned} \langle \textit{ObservationPredicate} \rangle &::= \langle \textit{EventView} \rangle\_\langle \textit{EventInformation} \rangle \\ \langle \textit{EventView} \rangle &::= \texttt{current} \mid \texttt{event} \mid \texttt{start} \\ \langle \textit{EventInformation} \rangle &::= \ldots \mid \texttt{env} \end{aligned}$$

- Two ReSpecT guard predicates are introduced:

$$\langle \textit{GuardPredicate} \rangle ::= \ldots \mid \texttt{from\_env} \mid \texttt{to\_env}$$

ReSpecT Tuple Centres as Environment-aware Media II

Coordination policies: talking about environment *(contd.)*

In the overall, ReSpecT is extended towards environment change

- by introducing some environment predicates and guards to get information about the environment status:

```
observation  current_env(?Key,?Value),
             start_env(?Key,?Value),
             event_env(?Key,?Value),
     guard   from_env, to_env
```

ReSpecT Tuple Centres as Environment-aware Media III

Capturing general environment events

2. A coordination medium should be able to capture environment events, and to react appropriately

- The ReSpecT *admissible (tuple centre) event* model is extended to include environment events

$$
\begin{array}{rcl}
\langle TCEvent \rangle & ::= & \langle OpEvent \rangle \mid \langle TEvent \rangle \mid \langle SEvent \rangle \mid \langle EEvent \rangle \ldots \\
\langle EEvent \rangle & ::= & \langle EStartCause \rangle , \langle EEventCause \rangle , \langle EResult \rangle , \\
 & & \langle Time \rangle , \langle Place \rangle , \langle Node \rangle \\
\langle EStartCause \rangle & ::= & \langle EOp \rangle , \langle EResId \rangle , \langle TCId \rangle \\
\langle EDirectCause \rangle & ::= & \langle EStartCause \rangle \\
\langle EOp \rangle & ::= & \texttt{env}(\langle Key \rangle , \langle Value \rangle) \\
\langle EResult \rangle & ::= & \langle EOp \rangle , \ldots
\end{array}
$$

ReSpecT Tuple Centres as Environment-aware Media IV

Capturing general environment events *(contd.)*

- Correspondingly, the ReSpecT *event descriptor* is extended, too

$$\begin{aligned}\langle Event\rangle \quad ::= \quad & \langle Predicate\rangle\,\texttt{(}\,\langle Tuple\rangle\,\texttt{)} \mid \texttt{time(}\,\langle Time\rangle\,\texttt{)} \mid \\ & \texttt{from(}\,\langle Place\rangle\,\texttt{)} \mid \texttt{to(}\,\langle Place\rangle\,\texttt{)} \mid \texttt{node(}\,\langle Node\rangle\,\texttt{)} \mid \\ & \texttt{env(}\,\langle Key\rangle\,\texttt{,}\,\langle Value\rangle\,\texttt{)}\end{aligned}$$

making it possible to specify reactions to the occurrence of environment events

```
reaction(env(?Key,?Value), Guard, Body).
```

Causing Environment Change I

Environment manipulation

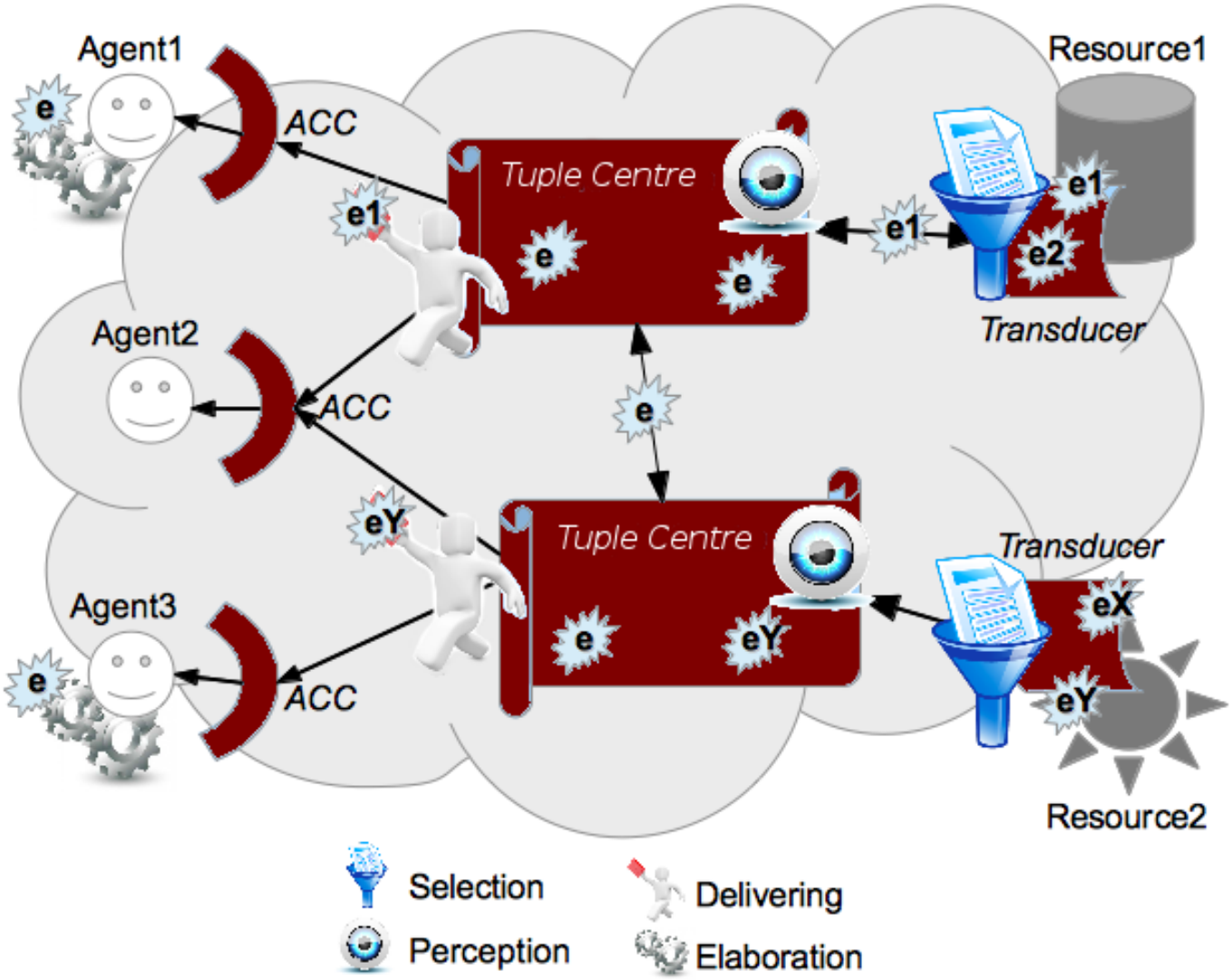
- Source and target of a tuple centre event can be any external resource
- A suitable *identification* scheme – both at the syntax and at the infrastructure level – is introduced for environmental resources
- The ReSpecT language is extended to express explicit manipulation of environmental resources
- The body of a ReSpecT reaction can contain a *tuple centre predicate* of the form
  - $\langle EResId \rangle$ `?` `get`($\langle Key \rangle$,$\langle Value \rangle$)
    enabling a tuple centre to get properties of environmental resources
  - $\langle EResId \rangle$ `?` `set`($\langle Key \rangle$,$\langle Value \rangle$)
    enabling a tuple centre to set properties of environmental resources

Causing Environment Change II

Transducers

- Specific environment events have to be translated into well-formed ReSpecT tuple centre events
- This should be done at the infrastructure level, through a general-purpose schema that could be specialised according to the nature of any specific resource
- A ReSpecT *transducer* is a component able to bring environment-generated events to a ReSpecT tuple centre (and back), suitably translated according to the general ReSpecT event model
- Each transducer is specialised according to the specific portion of the environment it is in charge of handling—typically, the specific resource it is aimed at handling, like a temperature sensor, or a heater.

ReSpecT-TuCSoN Architecture

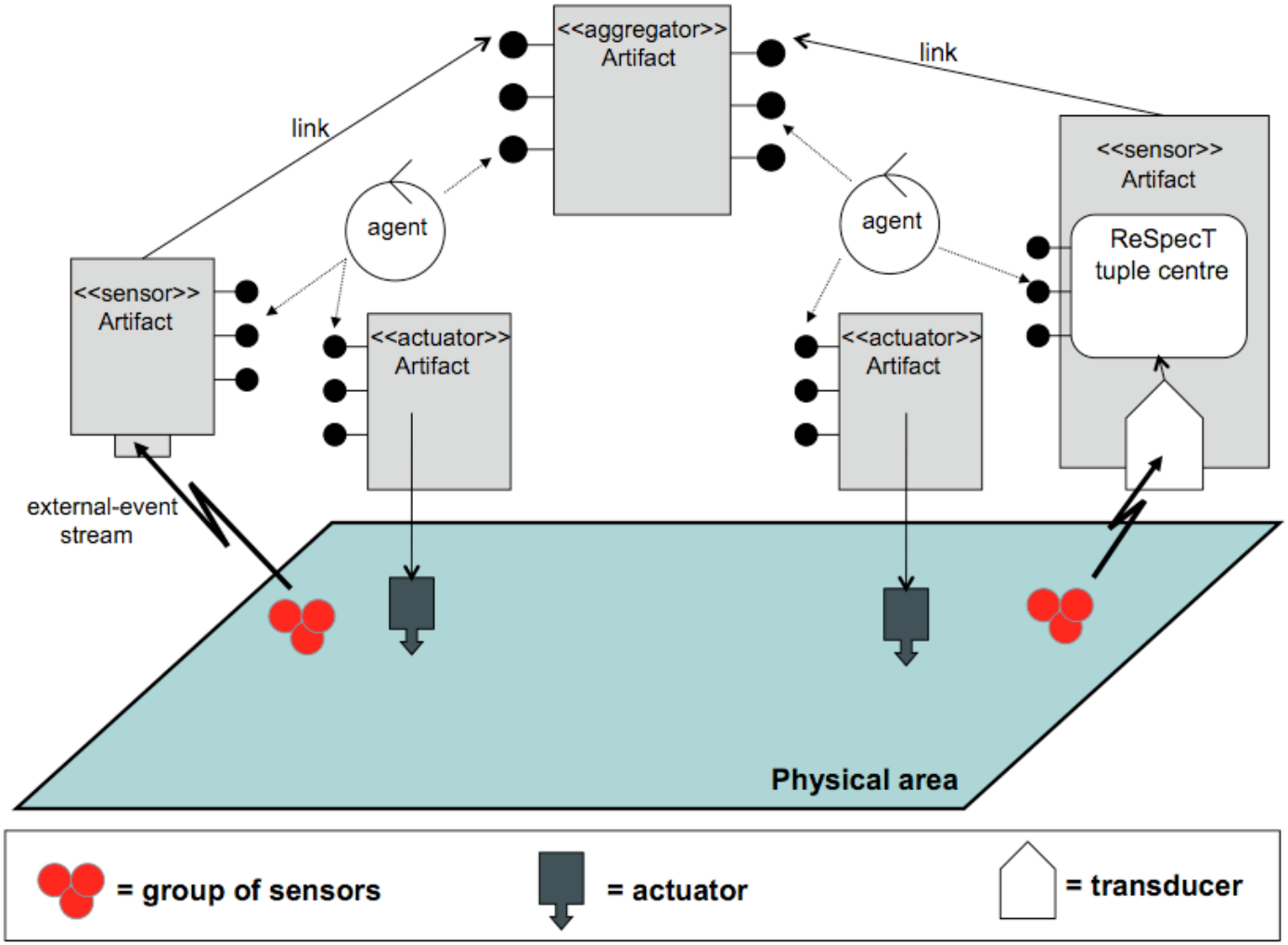


Outline

Controlling Environmental Properties of Physical Areas I

- A set of real *sensors* are used to measure some environmental property (for instance, temperature) within an area where they are located
- Such information is then exploited to govern suitably placed *actuators* (say, heaters) that can affect the value of the observed property in the environment
- Sensors are supposed to be cheap and non-smart, but provided with some kind of communication interface – either wireless or wired – that makes it possible to send streams of sampled values of the environmental property under observation
- Accordingly, sensors are active devices, that is, devices *pro-actively* sending sensed values at a certain rate with no need of being asked for such data—this is what typically occurs in *pervasive computing* scenarios
- Altogether, actuators and sensors are part of a distributed system aimed at controlling environmental properties (in the case study, temperature), which are affected by actuators based on the values measured by sensors and the designed control policies as well
- Coordination policies can be suitably automated and encapsulated within coordination media working as environment artifacts controlling sensors and actuators

Case Study: ReSpecT-based Architecture



Case Study: Structure of Environment Artifacts

Environment artifacts are built based on of ReSpecT tuple centres:

- `<<sensor>> artifacts` wrapping real temperature sensors which perceive temperature of different areas of the room
- `<<actuator>> artifacts` wrapping actuators, which act as heating devices so as to control temperature
- `<<aggregator>> artifact` provides an aggregated view of the temperature values perceived by sensors spread in the room since it is linked to `<<sensor>> artifacts`:
  - `<<sensor>> artifacts` update tuples on `<<aggregator>> artifact` through *linkability*

Case Study: Sensor Artifacts

```
%(1)
reaction( env(temperature, Temp), from_env, (
     event_time(Time), event_source(sensor(Id)),
     out(sensed_temperature(Id,Temp,Time)),
     tc_aggr@node_aggr ? out(sensed_temperature(Id,Temp)) )
).
%(2)
reaction( out(sensed_temperature(_,Temp,_)), from_tc, (
     in(current_temperature(_)),
     out(current_temperature(Temp)) )
).
```

Behaviour

- Reaction (1) is triggered by external events generated by a temperature sensor
- Reaction (2) updates current temperature

Case Study: Aggregator Artifacts

```
%(4)
reaction( out(sensed_temperature(Id,Temp)), from_tc, (
      in(total_temperature(OldTotalTemp),
      in(sensed_temperature(Id,OldTemp)),
      TotalTemp is OldTotalTemp - OldTemp + Temp,
      out(total_temperature(TotalTemp),
      rd(number_of_sensors(SensorNo),
      AvgTemp is TotalTemp / SensorNo,
      in(average_temp(_)), out(average_temp(AvgTemp)) )
).
```

Behaviour

- Reaction (4) keeps track of the current state of the average temperature

Case Study: Agents

Observable behaviour

Agents are goal-oriented and proactive processes that control temperature of the room

1. get local information from sensor
   `tc_sens@node_i ? rd(current_temperature(Temp_i))`
2. get global information from aggregator
   `tc_aggr@node_aggr ? rd(average_temp(AvgTemp))`
3. deliberate action by determining `TempVar` based on `Temp_i` and `AvgTemp`
4. act upon actuators (if `TempVar`$\neq 0$)
   `tc-heat_i@node_i ? out(change_temperature(TempVar))`

Case Study: Actuator Artifacts

```
%(3)
reaction( out(change_temperature(TempVar)), from_agent,
    actuator_i ? set(temp_inc,TempVar)
).
```

Behaviour

When the controller agent deliberate an increment in the temperature

- a `tc-heat_i@node_i ? out(change_temperature(TempVar))` reaches the actuator artifact
- by reaction (3), a suitable signal is sent to the actuator, through the suitably-installed transducer

Summing Up

Tuple-based models for Situated Coordination

- Situated coordination for mobile, pervasive, adaptive systems
- Situated coordination means
  - Time-aware coordination
  - Space-aware coordination
  - Environment-aware coordination
- Experiments of with ReSpecT

References I

Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg.
A survey on context-aware systems.
International Journal of Ad Hoc and Ubiquitous Computing, 2(4):263–277, 2007.

Jacob Beal.
A basis set of operators for space-time computations.
In *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW 2010)*, pages 91–97, Washington, DC, USA, 2010. IEEE Computer Society.

References II


Matteo Casadei and Andrea Omicini.
Situated tuple centres in ReSpecT.
In Sung Y. Shin, Sascha Ossowski, Ronaldo Menezes, and Mirko Viroli, editors, *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1361–1368, Honolulu, Hawai'i, USA, 8–12 March 2009. ACM.


Andrea Omicini.
Towards a notion of agent coordination context.
In Dan C. Marinescu and Craig Lee, editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA, October 2002.

References III


Andrea Omicini, Alessandro Ricci, and Mirko Viroli.
Time-aware coordination in ReSpecT.
In Jean-Marie Jacquet and Gian Pietro Picco, editors, *Coordination Models and Languages*, volume 3454 of *LNCS*, pages 268–282. Springer-Verlag, April 2005.
7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 April 2005. Proceedings.


Andrea Omicini, Alessandro Ricci, and Mirko Viroli.
Timed environment for Web agents.
Web Intelligence and Agent Systems, 5(2):161–175, August 2007.

References IV


Lucy A. Suchman.
Plans and Situated Actions: The Problem of Human-Machine Communication.
Cambridge University Press, New York, NYU, USA, 1987.


Danny Weyns, Andrea Omicini, and James Odell.
Environment as a first-class abstraction in multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):5–30, February 2007.
Special Issue on Environments for Multi-agent Systems.

References V

Franco Zambonelli, Gabriella Castelli, Laura Ferrari, Marco Mamei, Alberto Rosi, Giovanna Di Marzo Serugendo, Matteo Risoldi, Akla-Esso Tchao, Simon Dobson, Graeme Stevenson, Yuan Ye, Elena Nardini, Andrea Omicini, Sara Montagna, Mirko Viroli, Alois Ferscha, Sascha Maschek, and Bernhard Wally.

Self-aware pervasive service ecosystems.

Procedia Computer Science, 7:197–199, December 2011.

Proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11).

Tuple-based Coordination of Situated Systems

Distributed Systems

Sistemi Distribuiti

Andrea Omicini

andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)

Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015