Space-aware Coordination in ReSpecT

Stefano Mariani
DISI, ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, Italy
Email: s.mariani@unibo.it

Andrea Omicini
DISI, ALMA MATER STUDIORUM–Università di Bologna
via Sacchi 3, 47521 Cesena, Italy
Email: andrea.omicini@unibo.it

***Abstract*—Spatial issues are essential in new classes of complex software systems, such as pervasive, multi-agent, and self-organising ones. Understanding the basic mechanisms of spatial coordination is a fundamental issue for coordination models and languages in order to deal with such systems, governing situated interaction in the spatio-temporal fabric.**

Along this line, in this paper we make *space-aware coordination media* out of ReSpecT tuple centres, by introducing the few basic mechanisms and constructs that enable the ReSpecT language to face most of the main challenges of *spatial coordination* in complex software systems.

I. INTRODUCTION

Complex socio-technical systems in pervasive computing scenarios are stressing more and more the requirements for coordination middleware [1]. In particular, the availability of a plethora of mobile devices, with motion sensors and motion coprocessors, 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 [2]. In most of the application scenarios where *situatedness* plays an essential role, coordination is required to be *space aware*. This is implicitly recognised by a number of proposals in the coordination field trying to embody spatial mechanisms and constructs into the coordination languages – such as TOTA [3], $\sigma\tau$-LINDA [4], GEOLINDA [5], and SAPERE [1] – which, however, are mostly tailored around specific application scenarios.

In this paper we aim at devising out the *basic* mechanisms and constructs required to *generally* enable and promote *space-aware coordination*. Along this line, we introduce the general notion of space-aware coordination medium (Section II), then we show how the ReSpecT coordination media and language can be extended to support space-aware coordination (Section III). After sketching the semantics of the spatial extension (Section IV), Section V shows some meaningful examples of spatial coordination using ReSpecT. Section VI discusses related research, then Section VII provides for final remarks.

II. SPACE-AWARE COORDINATION MEDIA

A. Spatial Issues

Spatial coordination requires spatial *situatedness* and *awareness* of the coordination media, which translates in a number of technical requirements.

a) Situatedness: First of all, *situatedness* requires that a *space-aware coordination abstraction* should at any time be associated to an absolute positioning, both physical (i.e., the position in space of the computational device where the medium is being executed on) and virtual (i.e. the network node on which the abstraction is executed). If not a must-have, geographical positioning is also desirable, and quite a cheap requirement, too, given the widespread availability of mapping services nowadays.

More generally, this concerns both *position* and *motion* – every sort of motion –, which in principle include speed, acceleration, and all variations in the space-time fabric, also depending on the nature of space. In fact, software abstractions may move along a *virtual* space – typically, the network – which is usually *discrete*, whereas physical devices (robots, mobile devices) move through a *physical* space, which is mostly *continuous*; software abstractions, however, may also be hosted by mobile physical devices, and share their motion. As a result, a coordination abstraction may move through either a physical, continuous space, (e.g., I am in a given position of a tridimensional physical space) or a virtual, discrete space (e.g., I am on a given network node).

Physical positioning could be either *absolute* (say, I am currently at latitude X, longitude Y, altitude Z), *geographical* (I am in via Sacchi 3, Cesena, Italy), or *organisational* (I am in Room 5 of the DISI, site of Cesena). Absolute positioning is more or less always available in the days of mobile devices, usually through GPS services—which, coupled with mapping services, typically provide some sort of geographical positioning, too. Virtual positioning is available as a network service, and might be also labelled as either absolute (in terms of the node IP number, for instance) or relative (for instance, as a domain/subdomain localisation via DNS). Organisational location should be instead defined application- or middleware-level, and related to either absolute or virtual positioning.

Furthermore, a notion of *locality* may be available—so as to allow for the *local vs. global* dynamics which is typical of complex distributed systems such as pervasive ones. Locality could be strictly bound to positioning, but not necessarily so: being in the same position is not always the same as being in the same location.

b) Awareness: The main requirement of *spatial awareness* is that the ontology of a space-aware coordination medium should contain some notion of space. This means, first of all, that 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*. So, generally speaking, a

range of predicates / functions should be provided to access spatial information associated to any event occurring in the coordination medium (e.g., where the action causing the event took place, where the coordination medium is currently executing), and to perform simple computations over spatial information.

Also, space has to be embedded into the working cycle of the coordination medium: the event model should include *spatial events*, which affect coordination activity by triggering some space-related computation within the coordination abstraction. In fact, associating spatial information to generic events is not enough: space-related laws like "when at home, switch on the lights" cannot be expressed only referring to actions actually performed, but instead require a specialised notion of spatial event (such as "I am at home") to be triggered. So, a spatial event should be generated within a coordination medium, conceptually corresponding to changes in space—so, related to *motion*, such as starting from / arriving to a place. Spatial events should then be captured by the coordination medium, and used to activate space-aware coordination laws, within the normal working cycle of the coordination abstraction.

B. Spatial Tuple Centres

Tuple centres are introduced in TuCSoN [6] as coordination media meant at encapsulating any computable coordination policy within the coordination abstraction. Technically, a tuple centre is a *programmable* tuple space, i.e., a tuple space whose behaviour in response to events can be programmed so as to specify and enact any coordination policy [7], [8]. Tuple centres can then be thought as general purpose coordination abstractions, which can be suitably forged to provide specific coordination services. So, the core idea behind tuple centres is to have first-class coordination abstractions powerful enough to encapsulate and enforce at execution time the laws required to support coordination in complex distributed systems. This does not happen, for instance, in basic LINDA-like models [9], where complex coordination activities surpassing the limited expressive power of tuple-based coordination force spreading the global logic of coordination among individual agents [10].

In the same way as timed tuple centres empower tuple centres with the ability of embodying timed coordination laws [11], *spatial tuple centres* extend tuple centres so as to address the spatial issues depicted in the previous subsection.

First of all, the location a tuple centre is obtained through the notions of *current place*: which could be, for instance, the absolute position in space of the computational device where the coordination medium is being executed on, or the domain name of the TuCSoN node hosting the tuple centre, or the location on the map. Then, motion is conceptually represented by two sorts of spatial events: moving from a starting place, and stopping at an arrival place—in any sort of space / place.

With respect to the formal model defined in [12], this is achieved by extending the input queue of the environment events to become the multiset $SitE$ of time, environment, and spatial events, handled as input events by the *situation* transition ($\longrightarrow_s$)—as shortly discussed in Section IV. Whenever some motion of any sorts occurs (such as the physical device starting / stopping, or the node identifier changes), a spatial event is generated, and put in the tuple centre $SitE$ multiset, to be handled by the *situation* transition.

Then, analogously to operation and time events, it is possible to specify reactions triggered by spatial events—the so-called *spatial reactions*. Spatial reactions follow the same semantics of other reactions: once triggered, they are placed in the triggered-reaction set and then executed, atomically, in a non-deterministic order. As a result, a spatial tuple centre can be programmed to react to the motion either in physical or in virtual space, so as to enforce space-aware coordination policies.

Finally, a simple notion of locality is provided by the TuCSoN *node* abstraction: when coordination primitives are invoked with no node specification, they are handled as implicitly referring to the *local interaction space* hosted by the node; when a node identifier is instead associated to the invocation, then the primitive explicitly refers to the *global interaction space* [6].

III. SPATIAL ReSpecT

ReSpecT tuple centres are tuple centres based on first-order logic, adopted both for the communication language (logic tuples), and for the behaviour specification language (ReSpecT) [13]. Basically, reactions in ReSpecT are defined as Prolog-like facts of the form `reaction(Activity, Guards, Goals)` which specify the list of the operations (`Goals`) to be executed when a given event occurs (called *triggering event*, caused by an `Activity`) and some conditions on the event hold (`Guards` evaluate to true). Such operations make it possible to insert / read / remove tuples from the tuple space and the specification space of the tuple centre, but also to observe the properties of the triggering event, as well as to invoke operations over other coordination media. The core syntax of ReSpecT is shown in TABLE I.

According to the abstract model described in Subsection II-B, the ReSpecT language is extended to address spatial issues *(i)* by introducing some spatial predicates to get information about the spatial properties of both the tuple centre and the triggering event, and *(ii)* by making it possible to specify reactions to the occurrence of spatial events. The extension to the ReSpecT is shown in TABLE II.

In particular, the following *observation predicates* are introduced for getting spatial properties of triggering events within ReSpecT reactions:[1]

- `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

[1] A Prolog-like notation is adopted for describing the modality of arguments: + is used for specifying input argument, − output argument, ? input/output argument, @ input argument which must be fully instantiated.

⟨*Specification*⟩	::=	{ ⟨*Reaction*⟩ . }
⟨*Reaction*⟩	::=	reaction (⟨*Activity*⟩ [, ⟨*Guards*⟩] , ⟨*Goals*⟩)
⟨*Activity*⟩	::=	⟨*Operation*⟩ | ⟨*Situation*⟩
⟨*Operation*⟩	::=	out (⟨*Tuple*⟩) | (in | rd | no | inp | rdp | nop) (⟨*Template*⟩ [, ⟨*Term*⟩])
⟨*Situation*⟩	::=	time (⟨*Time*⟩) | env (⟨*Key*⟩ , ⟨*Value*⟩)
⟨*Guards*⟩	::=	⟨*Guard*⟩ | (⟨*Guard*⟩ { , ⟨*Guard*⟩ })
⟨*Guard*⟩	::=	request | response | success | failure | endo | exo | intra | inter | from_agent | to_agent | from_tc | to_tc | before (⟨*Time*⟩) | after (⟨*Time*⟩) | from_env | to_env
⟨*Goals*⟩	::=	⟨*Goal*⟩ | (⟨*Goal*⟩ { , ⟨*Goal*⟩ })
⟨*Goal*⟩	::=	[⟨*TupleCentre*⟩ ?] ⟨*Operation*⟩ | ⟨*EnvRes*⟩ (<- | ->) env (⟨*Key*⟩ , ⟨*Value*⟩) | ⟨*Observation*⟩ | ⟨*Computation*⟩ | (⟨*Goal*⟩ ; ⟨*Goal*⟩)
⟨*Observation*⟩	::=	⟨*Selector*⟩_⟨*Focus*⟩
⟨*Selector*⟩	::=	current | event | start
⟨*Focus*⟩	::=	(activity | source | target) (⟨*Term*⟩) | time (⟨*Term*⟩)

TABLE I. ReSpecT SYNTAX: CORE *(without forgeability, bulk, uniform predicates)*

⟨*Situation*⟩	::=	time (⟨*Time*⟩) | env (⟨*Key*⟩ , ⟨*Value*⟩) | from (⟨*Space*⟩ , ⟨*Place*⟩) | to (⟨*Space*⟩ , ⟨*Place*⟩)
⟨*Guard*⟩	::=	request | response | success | failure | endo | exo | intra | inter | from_agent | to_agent | from_tc | to_tc | before (⟨*Time*⟩) | after (⟨*Time*⟩) | from_env | to_env | at (⟨*Space*⟩ , ⟨*Place*⟩) | near (⟨*Space*⟩ , ⟨*Place*⟩ , ⟨*Radius*⟩)
⟨*Focus*⟩	::=	(activity | source | target) (⟨*Term*⟩) | time (⟨*Term*⟩) | place (⟨*Space*⟩ , ⟨*Term*⟩)
⟨*Space*⟩	::=	ph | ip | dns | map | org

TABLE II. SPATIAL EXTENSIONS TO ReSpecT *(only affected definitions shown)*

where the node position can be specified as either its absolute physical position (`S=ph`), its IP number (`S=ip`), its domain name (`S=dns`), its geographical location (`S=map`) – as typically defined by mapping services like Google Maps –, or its organisational position (`S=org`)—that is, a location within an organisation-defined virtual topology.

As an example, the reaction specification tuple

```
reaction( in(q(X)),
  ( operation, completion ),
    ( current_place(ph,DevPos),
      event_place(ph,AgentPos),
      out(in_log(AgentPos,DevPos,q(X))) )).
```

when executed, inserts a new tuple (`in_log/3`) with spatial information each time a TuCSoN agent retrieves a tuple of the form `q(_)` from the tuple centre: this implements a sort of *spatial log*, recording absolute positions of both the querying agent and the device hosting the tuple centre.

Also, the following *guard predicates* are introduced to select reactions to be triggered based on spatial event properties:

- `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`.

So, for instance, `near(dns,'apice.unibo.it',2)` succeeds when the tuple centre is currently executing on a device whose second-level domain is `.unibo.it`.

Reactions to spatial events are specified similarly to ordinary reactions, by introducing the following new 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, Goals).
reaction(to(?Space,?Place), Guards, Goals).
```

As a simple example, consider the following specification tuples (wherever `Guards` is omitted, it is by default `true`):

```
reaction( from(ph,StartP),
  ( current_time(StartT)
    out(start_log(StartP,StartT)) )).
reaction( to(ph,ArrP),
  ( current_time(ArrT)
    out(stop_log(ArrP,ArrT)) )).
reaction( out(stop_log(ArrP,ArrT)),
  ( internal, completion ),
  ( in(start_log(StartP,StartT))
    in(stop_log(ArrP,ArrT))
    out(m_log(StartP,ArrP,StartT,ArrT)) )).
```

which altogether record a simple *physical motion log*, including start / arrival time and position. In fact, the first reaction stores information about the beginning of a physical motion in a `start_log/2` tuple, the second the end of the motion

in a `stop_log/2` tuple, whereas the last removes both sort of tuples and records their data altogether in a `m_log/4` tuple, representing the essential information about the whole trajectory of the mobile device hosting the tuple centre.

IV. Semantics

The basic ReSpecT semantics was first introduced in [13], then extended towards time-aware coordination in [11], reshaped to support the notion of coordination artefact in [8], finally enhanced with situatedness in [12]—which represents the reference semantics for ReSpecT till now.

In order to formalise the semantics for the space-aware extension of ReSpecT, two are the main changes with respect to [12]. First of all, a new, generalised event model should be defined to include both spatial events, and spatial information for any sort of event. Then, the *environment* transition, already handling both time and general environment events, should be extended to include spatial events. All other required extensions (such as the formalisation of each spatial construct's semantics) are technically simple, and trivially extend tables in [12].

A. Event Model

The first fundamental extension to the event model is clearly depicted in TABLE II: a new sort of *spatial* ⟨*Activity*⟩ is introduced. In particular, the notion of ⟨*Situation*⟩ is extended with the two spatial activities `from(` ⟨*Space*⟩ ⟨*Place*⟩ `)`, `to(` ⟨*Space*⟩ ⟨*Place*⟩ `)`, reflecting the initial and final stages of a motion trajectory, respectively—in whatever sort of space.

However, spatial extension of the event model cannot be limited to introducing spatial activities: another issue is represented by *spatial qualification* of events, that is, in short, making *all* ReSpecT events featuring spatial properties—in the same way as temporal properties were introduced for all ReSpecT events in [11]. This is represented by the ⟨*Place*⟩ property featured by ⟨*Cause*⟩ – and ⟨*StartCause*⟩, of course –, as shown in TABLE III, where the extended ReSpecT event model is depicted. Essentially, all ReSpecT events are in principle qualified with both time and space properties—the latter one defined as the position (in whichever sort of space) where the (initial) cause of the event takes place. Of course such properties may be defined or not, depending on the facility available when the event is generated. For instance, if absolute physical positioning is made available by the hosting device, and the device is currently in location `P` when an event is generated, the coordination middleware associates `P` to the event as its physical location—which otherwise would be left undefined.

B. Transition

According to [12], the operational semantics of a ReSpecT tuple centre is expressed by a transition system over a state represented by a labelled triple ${}^{OpE,SitE}\langle Tu, Re, Op\rangle_n^{OutE}$—abstracting away from the specification tuples Σ, which are not of interest in this paper. In particular, Tu is the multiset of the ordinary tuples in the tuple centre; Re is the multiset of the triggered reactions waiting to be executed; Op is the multiset of the requests waiting for a response; OpE is the multiset of incoming ⟨*Operation*⟩ events; $SitE$ is the multiset of incoming ⟨*Situation*⟩ events, including time, spatial, and general environment events; $OutE$ is the outgoing event multiset; n is the local tuple centre time.

$OutE$ is automatically emptied by emitting the outgoing events, with no need for special transitions. In the same way, OpE and $SitE$ are automatically extended whenever a new incoming (either operation or situation) event enters a tuple centre—again, no special transitions are needed for incoming events. In particular, $SitE$ is added new environment events by the associated transducers [12], new time events by the passing of time [11], and – in the spatial extension of ReSpecT presented here – also new spatial events whenever some sort of motion takes place.

So, as described in [12], the behaviour of a ReSpecT tuple centre is modelled by a transition system composed of four different transitions: *reaction* ($\longrightarrow_r$), *situation* ($\longrightarrow_s$), *operation* ($\longrightarrow_o$), *log* ($\longrightarrow_l$). Quite intuitively, spatial events are handled – in the same way as time and environment events – by the *situation* transition, which triggers ReSpecT reactions in response to spatial events. As a result, the *situation* transition is the fundamental, and now finally complete, ReSpecT machinery supporting situatedness in the full acceptation of the term—that is, suitably handling reactiveness of the coordination abstraction to time, space, and general environment events.

V. Examples

A. Spheric Broadcasting

In order to demonstrate the simplicity and effectiveness of the new ReSpecT spatial features, in the following we show the ReSpecT implementation of a sort of communication pattern, widely diffused in nature-inspired systems [14]: *spheric broadcasting*. There, a message has to be *locally spread* in the environment, so to be perceived only by nearby agents. In principle, the distance defining such "diffusion neighbourhood" can be thought of as a "straight-line" radius, identifying a three-dimensional sphere around the point where the message is firstly created. Nevertheless, if this message is, e.g., a digital pheromone left by an ant-like agent, other properties may affect message broadcasting: such as time, for instance, making the message unavailable after a while—which can be programmed in ReSpecT, too, thanks to its timed extension [11].

Since we are mainly concerned with the gain in expressiveness provided by the spatial extension to ReSpecT here proposed, the following ReSpecT specification is focussed on spatial-sensitivity only, and can be logically split into two parts: Reactions `1.1`, `1.2` manage spheric broadcasting requests, whereas Reactions `2.1`, `2.2` enact the spreading process.

```
% 1) Get agent message
% 1.1) Delete garbage tuple
reaction( out(spheric(Msg,Radius)), completion,
  in(spheric(Msg,Radius)) ).
% 1.2) Check range then forward
reaction( out(spheric(Msg,Radius)), completion,
  ( current_place(ph,RecPos),           % Current position
    start_place(ph,SendPos),            % Starting position
    in_range(ph,RecPos,SendPos,Radius), % Prolog computation
    start_source(Sender), start_time(Time),
    out(msg(Msg,Sender,Time)),
```

⟨*Event*⟩	::=	⟨*StartCause*⟩ , ⟨*Cause*⟩ , ⟨*Evaluation*⟩
⟨*StartCause*⟩ , ⟨*Cause*⟩	::=	⟨*Activity*⟩ , ⟨*Source*⟩ , ⟨*Target*⟩ , ⟨*Time*⟩ , ⟨*Space:Place*⟩
⟨*Source*⟩ , ⟨*Target*⟩	::=	⟨*AgentId*⟩ | ⟨*TCId*⟩ | ⟨*EnvResId*⟩ | ⊥
⟨*Evaluation*⟩	::=	⊥ | {⟨*Result*⟩}
⟨*Place*⟩	::=	⟨*GPSCoordinates*⟩ , ⟨*IPAddress*⟩ , ⟨*DomainName*⟩ , ⟨*MapLocation*⟩ , ⟨*VirtualPosition*⟩

TABLE III. EXTENDING ReSpecT EVENTS WITH SPACE

```
    rd(neighbours(Nbrs)),            % List of neighbours
    out(forward(Nbrs,Msg,Radius)) )).
% 2) Forward to every neighbour
% 2.1) Delete garbage tuple
reaction( out(forward(Nbrs,Msg,Radius)),
  ( internal, completion ),
  in(forward(Nbrs,Msg,Radius)) ).
% 2.2) Neighbour list not empty
reaction( out(forward([Nbr|Nbrs],Msg,Radius)),
  ( internal, completion ),
  ( Nbr ? out(spheric(Msg,Radius)),  % Forward
    out(forward(Nbrs,Msg,Radius)) )).% Iterate
```

More specifically,

- Reaction `1.2` reacts to spheric broadcasting requests:
  - by acquiring spatial information about the physical position of the ReSpecT tuple centre currently managing the request, as well as about the entity – either an agent or another tuple centre – sending the request;
  - by checking if current tuple centre is within the given `Radius` w.r.t. the sender's position;
  - if so, it actually stores the message – along with some contextual information – then starts the spreading process;
  - if not so, ReSpecT transactional semantics makes the whole reaction fail, rollbacking all changes made (if any).
- Reaction `1.1` simply consumes the request tuple—no longer needed both in case of `in_range/3` predicate success and failure.
- Reaction `2.2` exploits the *linkability* feature of ReSpecT tuple centres to forward spheric broadcast request to each neighbour, iteratively.
- Reaction `2.1` simply removes the forwarding tuple when all neighbours have been considered.

Given a virtual topology is reified in the tuple `neighbours` – e.g. resembling physical network links between nodes – and the above ReSpecT specification is installed on every tuple centre of a network, we get that any message embedded in a `spheric` tuple is autonomously spread *solely* to "`Radius`-reachable" neighboring tuple centres. This is made possible by the `start_place/2` observation predicate, available in every tuple centre to inspect the place where the *first* spheric broadcast request was issued, thus enabling (physical) range check regardless of the number of tuple centre hops taken to forward the request.

Furthermore, one should note how completely different communication patterns could be obtained through small modifications of the ReSpecT code above:

- by replacing the space qualifier `ph` with `ip` or `dns`, one could achieve, e.g., *subnet broadcasting*, that is, a broadcast communication limited to a given subnet, where such a subnet can be arbitrarily computed through the `in_range/4` predicate.
- by replacing event view predicate `start` with `event`, one could achieve, e.g., *neighborhood-driven, global broadcast*, that is, a broadcast communication covering the whole network, where each node forwards messages to its (spheric) neighborhood solely, but recursively. This is made possible by the fact that `event` considers the *direct cause* of the event, not the original one – as `start` does –, hence predicate `in_range/4` is in turn affected. Nevertheless, suitable ReSpecT reactions should be added to avoid flooding.

This should illustrate the expressive power conveyed by every single predicate: changing even a few of them in a ReSpecT reaction has the potential to strongly impact on the semantics of a ReSpecT specification—and then, on the coordinative behaviour of a ReSpecT tuple centre.

B. Monitored Motion

The second example, while focussing on the expressiveness gain given by ReSpecT spatial extension like the previous one, is also meant to whow how the different features of ReSpecT – space-time awareness, situatedness, and programmability – can be combined to cope with complex problems, such as *monitored motion*.

We call "monitored motion" the problem in which a mobile device – e.g. a simple wheel-equipped robot – has to reach a given destination, which implies *(i)* to start moving when asked to do so; *(ii)* to monitor its own position so as to understand when given destination is reached, if obstacles are on its way, etc.; *(iii)* to finally stop when due. In the following ReSpecT specification – where the three reactions resemble the three different stages composing monitored motion – we explicitly monitor arrival to destination only, for the sake of simplicity.

More specifically, looking at the code below

- Reaction `1` reacts to movement requests by:
  - recording current position to compute the "movement vector" to follow, based on given destination;
  - sampling current time to schedule a monitoring reaction when specified in `TStep`—exploiting time awareness;
  - setting environmental properties, in particular, direction of motion and engine state of a `motion_dev` actuator on the hosting device (`Node`)—exploiting situatedness.
- Reaction `2` is responsible of motion monitoring:

- first (predicate `check/2`), it perceives current position and given destination to understand if controlled device is arrived;
- if `check/2` returns `true`, the motion engine is turned off so to stop;
- if `check/2` returns `false`, the reaction re-schedules itself to continue monitoring.

- Reaction `3` does some clean-up when destination is reached, and spatial event `to(Dest)` is generated.

```
% 1) Compute motion vector then start moving.
  reaction( out(move(Dest,TStep)), completion,
    ( current_place(ph,Here), current_time(Now),
      Check is Now+TStep,
      direction(Dest,Here,Vec),    % Prolog computation
      out_s( % Reaction #2 ),      % Schedule monitoring
      current_target(_@Node),      % Get node id
      motion_dev@Node <- env(dir,Vec),   % Set direction
      motion_dev@Node <- env(engine,'on') )).  % Move on
% 2) Monitor destination arrival.
  reaction( time(Check), internal, % Time to check
    ( current_place(ph,Here),
      rd(move(Dest,TStep)),
      ( check(Here,Dest),           % Prolog computation
        current_node(Node),
        motion_dev@Node <- env(engine,'off')
        ;                           % 'if-then-else'
        current_time(Now), Check is Now+TStep,
        out_s( % Reaction #2 ) ) )).
% 3) Arrival clean-up.
  reaction( to(Dest),               % Destination reached
    internal, in(move(Dest,_)) ).
```

VI. Related Works

The need for coordination models and languages to support adaptiveness and reactiveness to the contingencies that may occur in the spatio-temporal fabric was recognised by several proposals in the literature, accounting for spatial issues and/or enforcing spatial properties.

$\sigma\tau$-LINDA [4] extends the LINDA model in three ways to enable the emergence of spatio-temporal patterns for tuples configuration in a distributed setting:

- tuples are replaced by "space-time activities", that is, processes manipulating the space-time configuration of tuples in the network;
- space operators `neigh`, `$distance` and `$orientation` are added to allow respectively to send broadcast messages to the neighborhood spaces, measure the distance toward any of them, devise the relative orientation of a target space w.r.t. the current one;
- time operators `next`, `$delay` and `$this` are added allowing (respectively) to delay an activity execution, measure the flow of time, access current coordination space identifier.

GEOLINDA [5], another LINDA extension, deals with spatial aspects by attempting to reflect physical spatial properties in a mobile tuple space setting:

- each tuple and each reading operation is associated to a geometrical volume (*addressing shape*);
- the semantics of reading operations is changed so as to unblock only when the shape of a matching tuple intersects with the addressing shape of the operation;
- each coordination space is associated to a communication range to allow detection of *incoming* and *outgoing* tuples/volumes.

The TOTA middleware [3] considers a distributed tuple space setting in which each tuple is equipped by two additional fields other than its content:

- a *propagation rule*, determining how the tuple should propagate and distribute across the network of linked tuple spaces—e.g. in terms of maximum number of hops, conditional rules upon the presence of other tuples, even how the tuple can change while moving
- a *maintenance rule*, dictating how the tuple should react to the flow of time and/or to spatial events

The combination of these rules makes it possible for TOTA to support self-organising *computational fields*, that is, distributed data structures – such as gradients – enforcing spatial properties in the configuration of tuples, eventually exploited by coordinating agents.

Finally, the SAPERE coordination model [1] is a chemical-inspired model for pervasive applications, enacting spatial computing patterns and gradient-based interaction [15].

Although the above coordination models deal with some spatial-related issues, they are mostly tailored to a specific problem or application domain, and lack of an exhaustive analysis of which are the basic mechanisms required to enable and promote "general-purpose" space-aware coordination.

VII. Conclusion & Future Works

In this paper we take the ReSpecT coordination media and language [8], and extend it with few basic constructs and mechanisms required to build space-aware coordination media able to deal with spatial issues.

Future work will be devoted to explore real-world case studies to put to test both the expressiveness and the effectiveness of the space-aware ReSpecT model in the coordination of complex systems, such as pervasive and adaptive ones. For instance, the recent availability of the TuCSoN middleware [16], exploiting ReSpecT tuple centres, upon Android devices makes it possible to actually experiment with spatial coordination in pervasive scenarios.

Acknowledgments

This work has been partially supported by the EU-FP7-FET Proactive project SAPERE – Self-aware Pervasive Service Ecosystems, under contract no. 256874.

References

[1] F. Zambonelli, G. Castelli, L. Ferrari, M. Mamei, A. Rosi, G. Di Marzo Serugendo, M. Risoldi, A.-E. Tchao, S. Dobson, G. Stevenson, Y. Ye, E. Nardini, A. Omicini, S. Montagna, M. Viroli, A. Ferscha, S. Maschek, and B. Wally, "Self-aware pervasive service ecosystems," *Procedia Computer Science*, vol. 7, pp. 197–199, Dec. 2011, proceedings of the 2nd European Future Technologies Conference and Exhibition 2011 (FET 11). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050911005667

[2] J. Beal, O. Michel, and U. P. Schultz, "Spatial computing: Distributed systems that take advantage of our geometric world," *ACM Trans. Auton. Adapt. Syst.*, vol. 6, no. 2, pp. 11:1–11:3, Jun. 2011.

[3] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications: The tota approach," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, no. 4, pp. 15:1–15:56, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1538942.1538945

[4] M. Viroli, D. Pianini, and J. Beal, "Linda in space-time: an adaptive coordination model for mobile ad-hoc environments," in *Coordination Languages and Models*, ser. LNCS, M. Sirjani, Ed. Springer-Verlag, 14–15 Jun. 2012, vol. 7274, pp. 212–229, 14th Conference (Coordination 2012), Stockholm (Sweden).

[5] J. Pauty, P. Couderc, M. Banatre, and Y. Berbers, "Geo-Linda: a geometry aware distributed tuple space," in *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, may 2007, pp. 370 –377.

[6] A. Omicini and F. Zambonelli, "Coordination for Internet application development," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 3, pp. 251–269, Sep. 1999, special Issue: Coordination Mechanisms for Web Agents. [Online]. Available: http://springerlink.metapress.com/content/uk519681t1r38301/

[7] A. Omicini and E. Denti, "From tuple spaces to tuple centres," *Science of Computer Programming*, vol. 41, no. 3, pp. 277–294, Nov. 2001.

[8] A. Omicini, "Formal ReSpecT in the A&A perspective," *Electronic Notes in Theoretical Computer Science*, vol. 175, no. 2, pp. 97–117, Jun. 2007, 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 Aug. 2006. Post-proceedings. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571066107003519

[9] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, Jan. 1985. [Online]. Available: http://portal.acm.org/citation.cfm?id=2433

[10] E. Denti, A. Natali, and A. Omicini, "Programmable coordination media," in *Coordination Languages and Models*, ser. LNCS, D. Garlan and D. Le Métayer, Eds. Springer-Verlag, 1997, vol. 1282, pp. 274–288, 2nd International Conference (COORDINATION'97), Berlin, Germany, 1–3 Sep. 1997. Proceedings. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-63383-9_86

[11] A. Omicini, A. Ricci, and M. Viroli, "Time-aware coordination in ReSpecT," in *Coordination Models and Languages*, ser. LNCS, J.-M. Jacquet and G. P. Picco, Eds. Springer-Verlag, Apr. 2005, vol. 3454, pp. 268–282, 7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 Apr. 2005. Proceedings. [Online]. Available: http://www.springerlink.com/content/kbcmbqed8jta590g/

[12] M. Casadei and A. Omicini, "Situated tuple centres in ReSpecT," in *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, S. Y. Shin, S. Ossowski, R. Menezes, and M. Viroli, Eds., vol. III. Honolulu, Hawai'i, USA: ACM, 8–12 Mar. 2009, pp. 1361–1368. [Online]. Available: http://portal.acm.org/citation.cfm?id=1529282.1529586

[13] A. Omicini and E. Denti, "Formal ReSpecT," *Electronic Notes in Theoretical Computer Science*, vol. 48, pp. 179–196, Jun. 2001, declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 Dec. 2000.

[14] A. Omicini, "Nature-inspired coordination models: Current status, future trends," *ISRN Software Engineering*, vol. 2013, 2013, article ID 384903, Review Article. [Online]. Available: http://www.hindawi.com/isrn/se/2013/384903/

[15] M. Viroli, M. Casadei, S. Montagna, and F. Zambonelli, "Spatial coordination of pervasive services through chemical-inspired tuple spaces," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 6, no. 2, pp. 14:1–14:24, Jun. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1968513.1968517

[16] TuCSoN, "Home page," `http://tucson.unibo.it/`, 2013.