Promoting Space-Aware Coordination: ReSpecT as a Spatial Computing Virtual Machine

Stefano Mariani, Andrea Omicini
{s.mariani, andrea.omicini}@unibo.it

Alma Mater Studiorum—Università di Bologna

SCW 2013
SCW @ AAMAS Conference
St. Paul, 6th of May 2013

Outline

The Need For Situatedness

Nowadays software systems – especially *distributed*, *ubiquitous* and *pervasive* ones – increasingly call for some degree of situatedness, that is the capability of the system to *recognize* and *react* properly to changes in its environment—enabling adaptiveness, in the end.

Spatial Situatedness

One dimension of such situatedness is along the "spatial axis", which basically amounts to being able of measuring some spatial properties, manipulating them performing some computation, then affecting those/others spatial properties in turn.

Situatedness Into Programming Languages

Spatial Computing Languages (SCL) born to address the issue of bringing situatedness *into* programming languages, allowing programmers to explicitly deal with space-related aspects *at the language level* [Beal et al., 2012].

Also Coordination Models recognized the importance of spatial situatedness, hence proposed some extensions to well-known languages to address the issue—notably [Viroli et al., 2012] and [Mamei and Zambonelli, 2009].

An Integrated Effort

Our goal here, is to share our effort in designing a Spatial Computing Virtual Machine on top of a Space-Aware Coordination Language, so as to (hopefully) stimulate new ideas and (possibly) converge to common solutions.

Outline

The Abstract Device Model

In [Beal et al., 2012] the Abstract Device Model (ADM) is defined so as to allow comparison between different SCL along three basic criteria:

Communication Region — The spatial "coverage" of the communication—e.g. global vs. neighbourhood

Communication Granularity — The number of receivers—e.g. unicast vs. multicast

Code Mobility — The relationships between different devices' running code

SCL Requirements

Furthermore, authors of [Beal et al., 2012] pointed out the need for three classes of operators in SCL to achieve a kind of *"Spatial-Turing-equivalence"* [Beal, 2010]:

Measure Space — To translate spatial properties into computable information—e.g. sensing GPS position

Manipulate Space — To translate back information into modifications of spatial properties—e.g. starting a motion engine

Compute (on) Space — Any kind of "spatial-pointwise" computation

A fourth class (Physical Evolution) looks more like an assumption about the dynamics that a given program/device must/can consider—e.g. a robot may move meanwhile a computation runs.

Outline

Space-Aware Coordination I

Extensions to the LINDA tuple-based coordination model have been proposed in [Viroli et al., 2012] and [Mamei and Zambonelli, 2009] to deal with spatial-related issues:

$\sigma\tau$LINDA *(i)* replaces standard tuples with *space-time activities*, that is, processes manipulating the space-time configuration of tuples in the network and *(ii)* adds to the model new space operators like `neigh`, `$distance` and `$orientation`, in order to allow management of spatial properties

TOTA equips each tuple with additional info: *(i)* a *propagation rule*, determining how the tuple should distribute across a network of tuple spaces and *(ii)* a *maintenance rule*, dictating how the tuple should react to the flow of time and/or events occurring in the space

Space-Aware Coordination II

On the one hand, these kind of works inspired us to reason about the opportunity of defining a *minimal* set of constructs needed to promote "general purpose" Space-Aware Coordination.

On the other hand, we believe one of the main concerns of SCL, that is, *"the ability to link the aggregate behaviour of a number of independent devices to the way in which they are individually programmed"* (from [Beal et al., 2012]), entails in the very end a coordination problem.

From Spatial -Computing to -Coordination

In fact, independently of the level of abstraction desired by a Spatial Computing programmer, "under the hood" its program should be somehow compiled/interpreted so as to *meaningfully (spatially) coordinate a set of entities*.

ReSpecT In One Slide

ReSpecT [Omicini and Denti, 2001a] is a Prolog-like language meant to program ReSpecT tuple centres [Omicini and Denti, 2001b], which are programmable tuple spaces.

"Programmable" means that ReSpecT allows to couple *coordination events* – e.g. a `in` request – with *computations*, to be be carried out *atomically* and *transactionally* by the ReSpecT Virtual Machine.

ReSpecT Reactions

`reaction(Event, Guards, Goals)`

- `Event` — Any coordination event involving the tuple centre
- `Guards` — Logic predicates to control reaction *triggering*
- `Goals` — ReSpecT computations to be carried out

ReSpecT Syntax Sketched

$$
\begin{array}{rcl}
\langle \mathit{SpecificationTuple} \rangle & ::= & \texttt{reaction(} \langle \mathit{TCEvent} \rangle \texttt{,} [\langle \mathit{Guard} \rangle \texttt{,}] \langle \mathit{Reaction} \rangle \texttt{)} \\
\langle \mathit{TCEvent} \rangle & ::= & \langle \mathit{TCPredicate} \rangle \texttt{(} \langle \mathit{Tuple} \rangle \texttt{)} \mid \\
 & & \texttt{time(} \langle \mathit{Time} \rangle \texttt{)} \mid \langle \mathit{EnvPredicate} \rangle \\
\langle \mathit{Reaction} \rangle & ::= & \langle \mathit{ReactionGoal} \rangle \mid \texttt{(} \langle \mathit{ReactionGoal} \rangle \{\texttt{,} \langle \mathit{ReactionGoal} \rangle\} \texttt{)} \\
\langle \mathit{ReactionGoal} \rangle & ::= & \langle \mathit{TCPredicate} \rangle \texttt{(} \langle \mathit{Tuple} \rangle \texttt{)} \mid \langle \mathit{ObsPredicate} \rangle \texttt{(} \langle \mathit{Tuple} \rangle \texttt{)} \mid \\
 & & \langle \mathit{TCLinkPredicate} \rangle \mid \langle \mathit{TCEnvPredicate} \rangle \\
\langle \mathit{TCPredicate} \rangle & ::= & \texttt{out} \mid \texttt{in} \mid \texttt{inp} \mid \texttt{rd} \mid \texttt{rdp} \mid \texttt{no} \mid \texttt{nop} \\
\langle \mathit{TCLinkPredicate} \rangle & ::= & \langle \mathit{TCId} \rangle \texttt{?} \langle \mathit{TCPredicate} \rangle \\
\langle \mathit{TCEnvPredicate} \rangle & ::= & \langle \mathit{EnvResId} \rangle \texttt{(<-|->)} \langle \mathit{EnvPredicate} \rangle \\
\langle \mathit{EnvPredicate} \rangle & ::= & \texttt{env(} \langle \mathit{Key} \rangle \texttt{,} \langle \mathit{Value} \rangle \texttt{)} \\
\langle \mathit{ObsPredicate} \rangle & ::= & \langle \mathit{EventView} \rangle\texttt{\_}\langle \mathit{EventInfo} \rangle \texttt{(} \langle \mathit{Tuple} \rangle \texttt{)} \mid \langle \mathit{EnvPredicate} \rangle \\
\langle \mathit{EventView} \rangle & ::= & \texttt{current} \mid \texttt{event} \mid \texttt{start} \\
\langle \mathit{EventInfo} \rangle & ::= & \texttt{predicate} \mid \texttt{tuple} \mid \texttt{source} \mid \texttt{target} \mid \\
 & & \texttt{time} \\
\langle \mathit{Guard} \rangle & ::= & \langle \mathit{GuardPredicate} \rangle \mid \texttt{(} \langle \mathit{GuardPredicate} \rangle \{\texttt{,} \langle \mathit{GuardPredicate} \rangle\} \texttt{)} \\
\langle \mathit{GuardPredicate} \rangle & ::= & \texttt{request} \mid \texttt{response} \mid \texttt{success} \mid \texttt{failure} \mid \ldots \\
 & & \texttt{before(} \langle \mathit{Time} \rangle \texttt{)} \mid \texttt{after(} \langle \mathit{Time} \rangle \texttt{)} \mid \\
 & & \texttt{from\_env} \mid \texttt{to\_env} \\
\ldots & \ldots & \ldots
\end{array}
$$

Outline

ReSpecT Device Model

Communication Region — Any tuple centre can communicate with any other network-reachable tuple centre through *linking operations* [Omicini et al., 2006] (⟨*TCLinkPredicate*⟩). The "basic" communication model of ReSpecT is thus *global communication*

Communication Granularity — No first-class coordination operations exist in ReSpecT allowing communication with a set of target tuple centres at once. Thus, ReSpecT naturally supports only *point-to-point communication*

Code Mobility — ReSpecT features meta-coordination operations dealing with specification tuples—similar to ⟨*TCPredicate*⟩. Therefore, ReSpecT tuple centres feature *code mobility*

*st*ReSpecT Language Extensions

Complementarily to the timed extension of ReSpecT, the following constructs makes a tuple centre space aware, that is, able to *recognize*, *measure* and – in synergy with [Casadei and Omicini, 2009] – *manipulate* events and properties belonging to the spatial dimension of a situated coordinated system:

- `from(`⟨*Place*⟩`)`, `to(`⟨*Place*⟩`)` — Triggering reactions to *spatial events*, such as leaving from and arriving at a given location
- ⟨*EventView*⟩`_place` — Providing access to spatial information for the triggering event
- `at(`⟨*Place*⟩`)`, `near(`⟨*Place*⟩`,`⟨*Radius*⟩`)` — Allowing for evaluation of the reaction goals when the given spatial conditions are met

*st*ReSpecT vs. SCL Requirements I

Measure Space — Three *Observation Predicates* are given:

- `current_place` — Measuring where the tuple centre executing the current *st*ReSpecT reaction is
- `event_place`, `start_place` — Measuring respectively where the *direct cause* and *prime cause* [Omicini, 2007] of the event triggering the current reaction took place

Manipulate Space — Addressed by the constructs of [Casadei and Omicini, 2009], e.g.:

- ⟨*EnvResId*⟩ `<- env(`⟨*Key*⟩, ⟨*Value*⟩`)` — Allowing an agent to dispatch commands to any device, modeled as an *environmental resource*

Compute (on) Space — Here falls standard ReSpecT computation, which can now rely on new "spatial" guards, e.g.:

- `near(`⟨*Place*⟩, ⟨*Radius*⟩`)` — `true` if the reacting tuple centre is near the given location

The "T-Program"

As a reference *benchmark* to compare the expressiveness of different SCL, the "T-Program" is proposed in [Beal et al., 2012] w.r.t. a set of independent moveable devices:

- i Cooperatively create a local coordinate system
- ii Self-organize into a "T"-shaped structure
- iii Determine its center of gravity, then draw a ring around it

SCL Requirements

Hence it requires all the three classes of operators afore-mentioned: stage *(i)* requires *measuring space* capabilities; stage *(ii)* requires *manipulating space* capabilities; stage *(iii)* requires both computational capabilities and, again, measuring capabilities.

Local Coordinate System I

Setting a local coordinate system basically amounts to:

i choosing an origin node

ii making it *spread* a “vector tuple” to neighbours

iii (recursively) making them increment such vector

iv forwarding it to neighbours

Thus, the basic mechanism needed at the VM level to enable such computation at the application level, is what we call neighborhood spreading, whose *st*ReSpecT code is sketched in next slide.

Local Coordinate System II

```
% Check range then forward.
reaction( out(nbr_spread(msg(Msg),nbr(Dist),req(ID))),
  ( completion, success ),
  ( no(req(ID)), out(req(ID)), % Avoid flooding
    current_place(Me), event_place(Sender),
    within(Me,Sender,Dist), % Prolog computation
    out(msg(Msg)),
    rd(nbrs(Nbrs)), % Neighbours list
    out(forward(Msg,Dist,req(ID),Nbrs))
)  ).

% Delete multicast request.
...

% Forward to every neighbour.
reaction( out(forward(Msg,Dist,req(ID),[H|T])), % Some Nbrs
  ( intra, completion ),
  ( H ? out(nbr_spread(msg(Msg),nbr(Dist),req(ID))), % Forward
    out(forward(Msg,Dist,req(ID),T)) % Iterate
)  ).

% Delete iteration tuple.
...
```

"T"-shaped Structure I

To arrange nodes (tuple centres) so as to form a "T-shaped" structure, we need to:

i define spatial constraints representing the T (how much "tall", "fat", etc.)

ii make every node move so as to satisfy them

Thus, the basic mechanism needed at the VM level is motion monitoring and control.

“T”-shaped Structure II

```
% Compute motion vector then start moving.
reaction( out(move(Constraints)),
  ( completion, success ),
  ( current_place(Here),current_time(Now),Check is Now+1000,
    direction(Constraints,Here,Vec), % Prolog computation
    out_s(
      % Reaction 13-23
    ),
    engine@motion <- env(engine,'on'),  % Start actuators
    steering@motion <- env(dir,Vec)
)  ).
% Motion constraints monitoring.
reaction( time(Check),
  internal,
  ( current_place(Here),
    rd(move(Constraints)),
    ( check(Here,Constraints), % Prolog computation
      engine@motion <- env(engine,'off')
    ;
      current_time(Now), Check is Now+1000,
      out_s(
        % Reaction 13-23
)  )  )  ).
% Arrival clean-up.
reaction( to(Dest),
  internal,
  in(move(Constraints))
).
```

Center of Gravity

To first compute the focal point (FC) of the "T-shape", then draw a sphere around it, we need two basic mechanisms, both similar to the *neighborhood spreading* previously shown:

- a *bidirectional* neighborhood spreading to collect replies to sent messages—allowing to aggregate all the node's coordinates and count them
- a *spherical multicast* to draw the ring pattern

VM Minimal API

Nevertheless, the code for spherical multicast is almost identical to neighborhood spreading, but for the use of observation predicate `start_place` instead of `event_place`. This replacement (almost) alone suffices to completely change the *spatial properties* of communication.

Outline

Conclusion

- We tried to bring some results from the Coordination field into the Spatial Computing perspective
- In particular, we presented the *st*ReSpecT space-aware extension to the ReSpecT language & virtual machine as a suitable ADM for SCL
- We then sketched how a *st*ReSpecT-based VM can meet "T-Program" benchmark requirements w.r.t. spatial operators

Further Works

Despite ReSpecT [Omicini, 2007] being freely available[a] as part of the TuCSoN middleware [Omicini and Zambonelli, 1999], a full functioning, general purpose implementation of *st*ReSpecT is still under development.

[a] See `http://tucson.apice.unibo.it` and [Omicini and Mariani, 2013]

Thanks to. . .

- ...everybody here for listening
- ...the SAPERE team for bringing me here [1]

[1]This work has been supported by the EU-FP7-FET Proactive project SAPERE *Self-aware Pervasive Service Ecosystems*, under contract no.256873.

Bibliography I

Beal, J. (2010).
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. IEEE Computer Society.

Beal, J., Dulman, S., Usbeck, K., Viroli, M., and Correll, N. (2012).
Organizing the aggregate: Languages for spatial computing.
CoRR, abs/1202.5509.

Casadei, M. and Omicini, A. (2009).
Situated tuple centres in ReSpecT.
In Shin, S. Y., Ossowski, S., Menezes, R., and Viroli, M., editors, *24th Annual ACM Symposium on Applied Computing (SAC 2009)*, volume III, pages 1361–1368, Honolulu, Hawai'i, USA. ACM.

Mamei, M. and Zambonelli, F. (2009).
Programming pervasive and mobile computing applications: The TOTA approach.
ACM Transactions on Software Engineering and Methodology, 18(4).

Bibliography II

Omicini, A. (2007).
Formal ReSpecT in the A&A perspective.
Electronic Notes in Theoretical Computer Science, 175(2):97–117.
5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006.
Post-proceedings.

Omicini, A. and Denti, E. (2001a).
Formal ReSpecT.
Electronic Notes in Theoretical Computer Science, 48:179–196.
Declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 December 2000.

Omicini, A. and Denti, E. (2001b).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.

Omicini, A. and Mariani, S. (2013).
The TuCSoN *Coordination Model & Technology. A Guide.*

Bibliography III

Omicini, A., Ricci, A., and Zaghini, N. (2006).
Distributed workflow upon linkable coordination artifacts.
In Ciancarini, P. and Wiklicky, H., editors, *Coordination Models and Languages*, volume 4038 of *LNCS*, pages 228–246. Springer.
8th International Conference (COORDINATION 2006), Bologna, Italy, 14–16 June 2006. Proceedings.

Omicini, A. and Zambonelli, F. (1999).
Coordination for Internet application development.
Autonomous Agents and Multi-Agent Systems, 2(3):251–269.
Special Issue: Coordination Mechanisms for Web Agents.

Viroli, M., Pianini, D., and Beal, J. (2012).
Linda in space-time: an adaptive coordination model for mobile ad-hoc environments.
In Sirjani, M., editor, *Coordination Languages and Models*, volume 7274 of *LNCS*, pages 212–229. Springer-Verlag.
Proceedings of the 14th Conference of Coordination Models and Languages (Coordination 2012),Stockholm (Sweden), 14-15 June.

Promoting Space-Aware Coordination: ReSpecT as a Spatial Computing Virtual Machine

Stefano Mariani, Andrea Omicini
{s.mariani, andrea.omicini}@unibo.it

Alma Mater Studiorum—Università di Bologna

SCW 2013
SCW @ AAMAS Conference
St. Paul, 6th of May 2013