

Event-driven Programming for Situated MAS with ReSpecT Tuple Centres

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

DISI
ALMA MATER STUDIORUM—Università di Bologna

ABC:MI 2013
Koblenz, Germany - 17 September 2013



- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



Why Event-driven Programming in MAS? I

- The *environment* has been already recognised as one of the foremost sources of complexity in MAS [Weyns et al., 2007]
- Nevertheless, accounting for it is mandatory if we are to build *situated MAS*—namely, able to deal with environment ever-changing nature
- So, the *artefact* abstraction was conceived to properly model & engineer such environment [Omicini et al., 2008]

Motivations

- But, how to reconcile agents' pro-activeness with environment's own while preserving their *autonomy*—uncoupling in control?
- And how to do so preserving also the *situated* nature of *interactions*?

Why Event-driven Programming in MAS? II

We answer similar questions by:

Goals

- providing an *event model* for situated MAS programming
- exploiting *tuple centres* as the architectural solution preserving agents as well as artefacts autonomy—thus, decoupling in control
- showcasing how a *coordination language* adopting such event model and executed within such tuple centres can be proficiently exploited to program situated MAS

In order to be more practical, the **ReSpecT** language and ReSpecT tuple centres [Omicini and Denti, 2001b]¹ will be taken as our reference.

¹Available within the TuCSon coordination infrastructure [Omicini and Zambonelli, 1999].

Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS**
- 3 Tuple Centres for Event-driven MAS Coordination
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



Artefacts to Model MAS Environment

The **Agent & Artefact** (A&A) meta-model adopts artefacts as the main abstraction for modelling and engineering general-purpose computational MAS environments [Omicini et al., 2008].

Environment artefact to promote situatedness

There, *environment* (or, resource) *artefacts* are the fundamental abstractions to deal with MAS situatedness, suitably dealing with environment events.

However, as a meta-model, A&A does not directly provide neither the required computational model nor the technology to build environment artefacts.

Tuple Centres as Coordination Artefacts

Among the many available computational models for *coordination artefacts*, tuple-based ones already proved their effectiveness in coping with complex MAS engineering [Omicini and Viroli, 2011].

From tuple spaces...

However, the semantics of coordination policies is strictly defined by the available coordination primitives, thus any coordination requirement not reducible to them should be charged upon the agents—which is unacceptable [Ciancarini et al., 2000].

Tuple Centres as Events Mediators

This is the main motivation behind the definition of tuple centres as a **programmable extension** of tuple spaces [Omicini and Denti, 2001b].

... through tuple centres...

Along with the communication space (the tuples exchanged by agents) a tuple centre provides a **coordination space**, storing specification tuples meant to change the behaviour of the coordination media in response to **events** of any sort.

Then, if programmability is exploited to capture, **model** and manage any event happening within a MAS, tuple centres can play the role of **events mediators**.

Tuple Centres as Environment Artefacts

Given that in the A&A meta-model the only sources of events can be either the agents or the artefacts, tuple centres can thus be able to mediate both agent-generated events as well as environment-generated ones.

... to environment artefacts

So, in the very end, we are now able to leverage tuple centres from being purely coordination artefacts to be also suitable for implementing *environment artefacts*.

But: how to **model** unpredictable, heterogenous, situated events? Can tuple centres offer something from the **control-uncoupling** standpoint?

Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination**
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination**
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



Tuple Centre Events

General *Event* model

$$\langle Event \rangle ::= \langle OpEvent \rangle \mid \langle EnvEvent \rangle \mid \langle STEvent \rangle$$

operation events — any event whose *prime cause* is an agent
(coordination) operation

environment events — any event whose prime cause is a *transducer*
[Casadei and Omicini, 2009]—that is, a component able to
bring environment-generated events within a tuple centre

spatio-temporal events — any event whose prime cause belongs to the
space-time fabric into which the tuple centre is immersed

Operation Events

OpEvent model

$$\langle OpEvent \rangle ::= \langle PrimeCause \rangle, \langle DirectCause \rangle, \langle Tuple \rangle$$

The distinction between **prime** and **direct** cause is due to the *linkability* feature of artefacts [Omicini et al., 2008]: in the case of tuple centres, this means they are able to invoke operations on each other².

Prime cause

$$\langle PrimeCause \rangle ::= \langle CoordOp \rangle, \langle AgentId \rangle, \langle TCId \rangle$$

Direct cause

$$\langle DirectCause \rangle ::= \langle CoordOp \mid LinkOp \rangle, \langle AgentId \mid TCId \rangle, \langle TCId \rangle$$

²Combined with *forgeability* (programmability), allows to build **distributed event chains** among networked tuple centres.

Environment Events

EnvEvent model

$$\langle EnvEvent \rangle ::= \langle EnvCause \rangle, \langle EnvCause \rangle, \langle env(Key, Value) \rangle$$

Environment events are always generated by the environment in the form of environmental property changes.

Prime and direct cause coincide

$$\langle EnvCause \rangle ::= \langle \leftarrow \mid \rightarrow env(Key, Value) \rangle, \langle EnvResId \rangle, \langle TCId \rangle$$

The arrow notation ($\leftarrow \mid \rightarrow$) stands respectively for getting (\leftarrow) or setting (\rightarrow) an environmental property—allowing to model both **sensors** and **actuators**, respectively.

Space-Time Events

STEvent model

$$\langle STEvent \rangle ::= \langle STCause \rangle, \langle STCause \rangle, \langle SOP \mid TOP \rangle$$

Spatio-temporal events are always generated by the spatio-temporal fabric intrinsic to any computational environment.

Again, a unique “cause”

$$\langle STCause \rangle ::= \langle SOP \mid TOP \rangle, \text{time} \mid \text{space}, \langle TId \rangle$$

Spatio-temporal awarness

$$\begin{aligned} \langle SOP \rangle &::= \text{from}(Place) \mid \text{to}(Place) \mid \text{node}(Node) \\ \langle TOP \rangle &::= \text{time}(Time) \end{aligned}$$

Whereas from/1 and to/1 events are related to physical motion, node/1 models virtual motion—e.g., getting a new IP address.

Events in the Space-Time Fabric

If tuple centres are *spatio-temporally situated*, then any generated event should be spatio-temporally situated as well.

Situated events

In particular, all “cause” specifications defined above should be extended by adding the following information:

$\langle Time \rangle, \langle Place \rangle, \langle Node \rangle$

This event model is the one adopted by the ReSpecT language, which is meant to program ReSpecT tuple centres.

Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination**
 - (ReSpecT) Event Model
 - **Events Handling in ReSpecT**
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



ReSpecT Specification Tuples

In short, given a tuple centre event e , a specification tuple $\text{reaction}(E, G, B)$ associates a reaction $R\theta$ to event e if

- $\theta = \text{mgu}(E, e)$ —where mgu is the first-order logic most general unifier
- guard $G\theta$ holds

Semantics

$$\mathcal{Z}_\sigma(e) ::= \biguplus_{r \in \sigma} z(e, r), \quad \text{where} \quad z(e, r) ::= \begin{cases} {}^e R\theta & \text{if } \theta = \text{mgu}(E, e) \wedge G\theta \\ \emptyset & \text{otherwise} \end{cases}$$

Triggered reactions $\in \mathcal{Z}_\sigma(e)$ are then executed **asynchronously** w.r.t. both agents and other tuple centres interactions, following a non-deterministic order and according to an **atomic, transactional** semantics³.

³The interested reader is forwarded to [Omicini and Denti, 2001a, Omicini, 2007], in particular about function \mathcal{E} .

ReSpecT Asynchronous Computational Model

Within (and between) ReSpecT tuple centres everything happens according to the *“invocation-completion”* semantics formalised in [Omicini and Denti, 2001b].

“Invocation-Completion” semantics

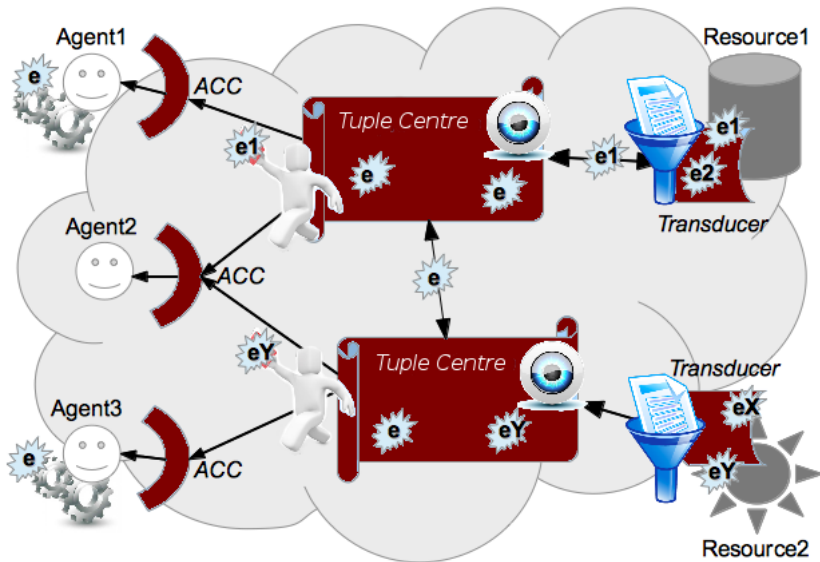
Whenever any operation is issued by whoever within a MAS^a two distinct events are generated:

- one in the *invocation phase*, that is when the operation is requested
- one in the *completion phase*, that is once the operation has been served

^aBe it an environmental resource, an agent, or a linking tuple centre.

This allows to manage any interaction *asynchronously*, leading to a full *uncoupling in control* between the interacting entities, thus preserving both agents and tuple centres *autonomy*.

ReSpecT-TuCSon Architecture



Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples**
- 5 Conclusion & Ongoing Work



Generative Communication, Logic & Inspectability I

- Suppose to have a MAS in which some form of “causal relation” must be enforced between (inter)actions performed by agents
 - e.g., insertion of some information is allowed only if other information already exists
- This amounts to verify some kind of “logical implication” between events happening within the MAS
 - e.g., allowing insertion of a tuple if and only if all “precondition tuples” have been already stored
- Similar issues can be dealt with by
 - adopting an event model suitable for **inspection** of all the needed information
 - exploiting an architecture promoting **generative communication** so as to preserve interacting entities autonomy
 - programming in a language supporting all these features—e.g., ReSpecT

Generative Communication, Logic & Inspectability II

```

1 reaction(out(Tuple),
2   (operation,completion),
3   (
4     start_source(Src), start_target(Trg),
5     causalCheck(event(out(Tuple),from(Src),to(Trg))) % Prolog
6     ; % 'if-then-else'
7     in(Tuple) % Remove inconsistency
8   )
9 ).
10
11 causalCheck(event(Op,F,T):-
12   findall(B, clause(Op,B), L), % Consult theory
13   causalCheck(F,T,L).
14
15 causalCheck(_,_, []).
16 causalCheck(F,T,[H|Tail]):-
17   rd(event(H,F,T), % Check consistency
18   causalCheck(F,T,Tail).

```


Situatedness & Linkability I

- Suppose to have a MAS in which a swarm of agents have to coordinate themselves so as to reach a uniform distribution in space
 - e.g., a swarm of robots moving in a building
 - e.g., an ensemble of web crawlers moving between subnets of the same network
- Suppose local communication should be used, as well as a distributed approach to the problem
 - e.g., robots are busy in a rescue mission after an earthquake, thus reliable, long-distance communications may be compromised
- Similar issues can be dealt with by
 - adopting an event model featuring **situatedness**
 - exploiting an architecture promoting **asynchronous communication** and distributed events handling
 - programming in a language supporting all these features—e.g., ReSpecT

Situatedness & Linkability II

```

1 reaction(out(moved(NewPos)),
2   (link_in,completion), % Incoming link
3   (
4     in(moved(NewPos)),
5     event_source(Who), % Direct cause inspection
6     in(nbr(Who, Pos)),out(nbr(Who,NewPos)),rd_all(nbr(_, _),Nbrs),
7     computeBaricenter(Nbrs,B), % Prolog computation
8     current_node(Host), % Inspect actual host
9     steering_engine@Host<-env('destination',B), % Command actuator
10    motion_engine@Host<-env('power','on') % Command actuator
11  )
12 ).
13 reaction(from(Pos), % Motion awareness
14   (internal),
15   (
16     % Avoid obstacles, collisions, etc.
17     % Monitor arrival to destination
18   )
19 ).
20 reaction(to(NewPos), % Stillness awareness
21   (internal),
22   (
23     rd_all(nbr(_, _),Nbrs),multicast(moved(NewPos),Nbrs)
24   )
25 ).
26 multicast(_, []).
27 multicast(Info,[nbr(N,_)|Nbrs]):-
28   N ? out(moved(Info)), % Outgoing link
29   multicast(Info,Nbrs).

```

Outline

- 1 Motivations & Goals
- 2 Artefacts, Coordination & Tuple Centres within MAS
- 3 Tuple Centres for Event-driven MAS Coordination
 - (ReSpecT) Event Model
 - Events Handling in ReSpecT
- 4 Event-driven Programming in ReSpecT: Examples
- 5 Conclusion & Ongoing Work



What Has Been Done

- The proposed event model has been implemented in TuCSoN [Omicini and Zambonelli, 1999]⁴, the coordination infrastructure featuring ReSpecT tuple centres
- ReSpecT language constructs have been extended so as to deal with situatedness-related issues
- The whole thing has been brought to Android, to benefit from mobile devices built-in hardware—e.g. GPS, accelerometer, gyroscope

⁴Publicly available under LGPL license from tucson.unibo.it.

To Do

- The porting is far from being complete, mostly due to many mobility-related issues still to be dealt with—e.g. low battery, low memory, transient network connection
- TuCSoN internal architecture needs to be improved so as to better adapt its capabilities to the hosting device it's running on—e.g. automatic detection of hardware support for situatedness

Thanks to everybody here at **ABC:MI 2013** for listening :)



References I



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.



Ciancarini, P., Omicini, A., and Zambonelli, F. (2000).

Multiagent system engineering: The coordination viewpoint.

In Jennings, N. R. and Lespérance, Y., editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, volume 1757 of *LNAI*, pages 250–259. Springer.







Omicini, A. (2007).



Formal ReSpecT in the A&A perspective.

Electronic Notes in Theoretical Computer Science, 175(2):97–117.

References II

-  Omicini, A. and Denti, E. (2001a).
Formal ReSpecT.
Electronic Notes in Theoretical Computer Science, 48:179–196.
-  Omicini, A. and Denti, E. (2001b).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.
-  Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17(3):432–456.
-  Omicini, A. and Viroli, M. (2011).
Coordination models and languages: From parallel computing to self-organisation.
The Knowledge Engineering Review, 26(1):53–59.

References III

-  Omicini, A. and Zambonelli, F. (1999).
Coordination for Internet application development.
Autonomous Agents and Multi-Agent Systems, 2(3):251–269.
-  Weyns, D., Omicini, A., and Odell, J. J. (2007).
Environment as a first-class abstraction in multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):5–30.



Event-driven Programming for Situated MAS with ReSpecT Tuple Centres

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

DISI
ALMA MATER STUDIORUM—Università di Bologna

ABC:MI 2013
Koblenz, Germany - 17 September 2013

