ReSpecT: **R**eaction **Spec**ification **T**uples
Advanced

Stefano Mariani
s.mariani@unibo.it

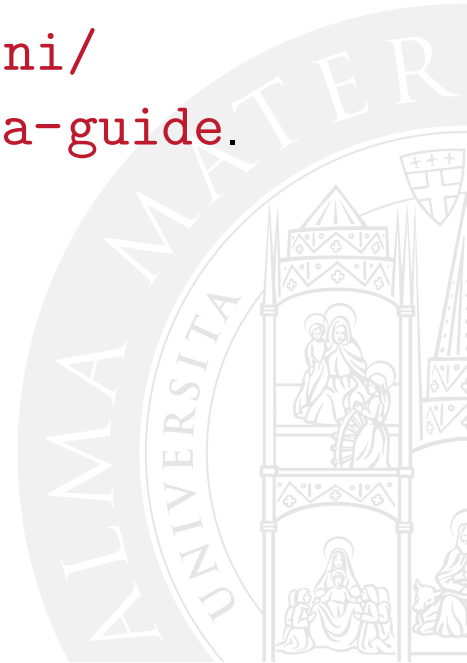
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015

1. ReSpecT Java APIs
2. ReSpecT Advanced Constructs
3. Bibliography

Disclaimer

These slides are adapted, arranged, integrated, starting from the official TuCSoN guide available at
http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide.



1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

External APIs

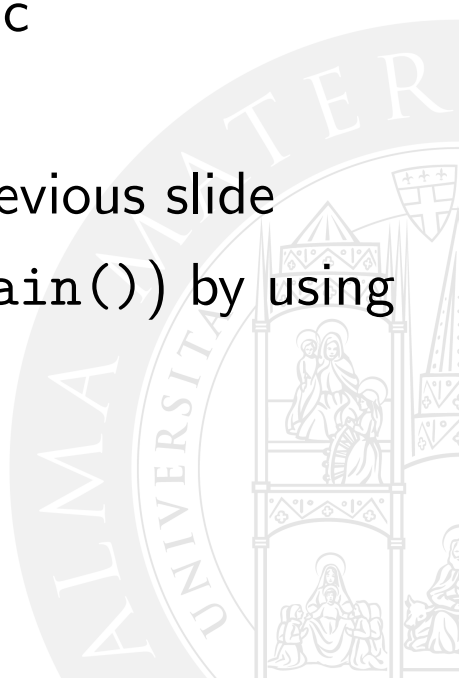
Uniform w.r.t. TuCSoN APIs accessing the ordinary tuples space:

1. build a `TucsonAgentId`
2. get a TuCSoN ACC allowing ReSpecT programming (e.g. `SynchACC` extends `SpecificationSynchACC`)
3. define the tuplecentre target of your meta-coordination operations
4. build a specification tuple for each construct of a ReSpecT reaction
   - → `LogicTuple event = LogicTuple.parse("out(t(X))");`
   - → `LogicTuple guards = LogicTuple.parse("(completion, success)");`
   - → `LogicTuple body = LogicTuple.parse("(out(in(t(X))), out(tuple(X)))");`
5. perform the meta-coordination operation using a meta-coordination primitive
   - → `ITucsonOperation op = acc.out_s(tid, event, guards, body, null);`
6. check requested operation success
7. get requested operation result

Extension APIs

Again, nothing new should be done except exploiting a suitable ACC:

1. extend `alice.tucson.api.TucsonAgent` base class
2. choose one of the given constructors, e.g.
3. override `main()` method with your agent business logic
4. get your ACC from the super-class
5. do what you want to do following steps 3 – 7 from previous slide
6. instantiate your agent and start its execution cycle (`main()`) by using method `go()`



Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

ReSpecT Events Revised

By recalling [Omicini, 2007], we may note that a ReSpecT event is not limited to be any ReSpecT (either coordination or meta-coordination) primitive:

$$\langle SimpleTCEvent \rangle ::= \langle SimpleTCPredicate \rangle \ (\langle Tuple \rangle) \mid \texttt{time}(\langle Time \rangle)$$

1. A `time(`T`)` event is generated by the ReSpecT VM when its *local time* (a relative notion of time measured starting from ReSpecT VM boot) reaches T
2. Then, any reaction having that time value as a triggering event is triggered and (iff guards evaluate to `true`) executed

ReSpecT Guards Revised

Correspondingly, other guard predicates are provided to manage time-related aspects:

`before(`*Time*`)` $\iff$ the triggering event occurred before *Time*

`after(`*Time*`)` $\iff$ the triggering event occurred after *Time*

`between(`*Time*`,`*Time'*`)` $\iff$ `after(`*Time*`)` & `before(`*Time'*`)`

Time-Aware Coordination Medium

- Time-awareness is an essential feature to enable situatedness, that is the ability of a system to recognize the *temporal* environment in which it lives, thus to react properly to its contingencies and dynamism
- the other fundamental feature is Space-awarness, that is the ability to recognize the *spatial* environment, properly expressing, generating and perceiving *topology-related* aspects

To learn more...

...please refer to the following papers

Timed ReSpecT [Omicini et al., 2005]

Situated ReSpecT [Casadei and Omicini, 2009]

Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

ReSpecT A&A Inspectability

Another extension to the original ReSpecT, provided by its A&A interpretation, is in ReSpecT events observability (*inspectability* of artifacts in the A&A terminology).

Observation predicates

In fact, the body of a ReSpecT reaction is not limited to exploit only ReSpecT primitives and Prolog predicates, but can use the so-called observation predicates:

$\langle \textit{ObservationPredicate} \rangle ::= \langle \textit{EventView} \rangle\_\langle \textit{EventInformation} \rangle$

$\langle \textit{EventView} \rangle ::= \texttt{current} \mid \texttt{event} \mid \texttt{start}$

$\langle \textit{EventInformation} \rangle ::= \texttt{predicate} \mid \texttt{tuple} \mid \texttt{source} \mid \texttt{target} \mid \texttt{time}$

Observability Semantics

Any combination of the following is admissible in ReSpecT, following formal grammar of ⟨*ObservationPredicate*⟩ in previous slide

⟨*EventView*⟩ — allow to inspect the *events chain* triggering the executing reaction:

- `current` — access the ReSpecT event currently under processing
- `event` — access the ReSpecT event which is the direct cause of the event triggering the reaction
- `start` — access the ReSpecT event which is the prime cause of the event triggering the reaction

⟨*EventInformation*⟩ — allow to inspect all the data ReSpecT events make observable:

- `predicate` — the ReSpecT primitive causing the event
- `tuple` — the logic tuple argument of the `predicate`
- `source` — who performed the `predicate`
- `target` — who is directed to the `predicate`
- `time` — when the `predicate` was issued

Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

Situating TuCSoN I

TuCSoN coordination for environment engineering

- Distributed systems are *situated*—that is, immersed into an environment, and reactive to events of *any* sort
- Thus, coordination media are required to mediate any activity toward the environment, allowing for a fruitful interaction

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

Situating TuCSoN II

Situating TuCSoN

- Thus, *situating* TuCSoN basically means making it capable of *capturing environment events*, and *expressing general MAS-environment interactions*
  [Casadei and Omicini, 2009, Omicini and Mariani, 2013]

$\Rightarrow$ the TuCSoN middleware and the ReSpecT language

- capture, react to, and observe general environment events
- explicitly interact with the environment

Dealing with Environment Change I

Environment manipulation

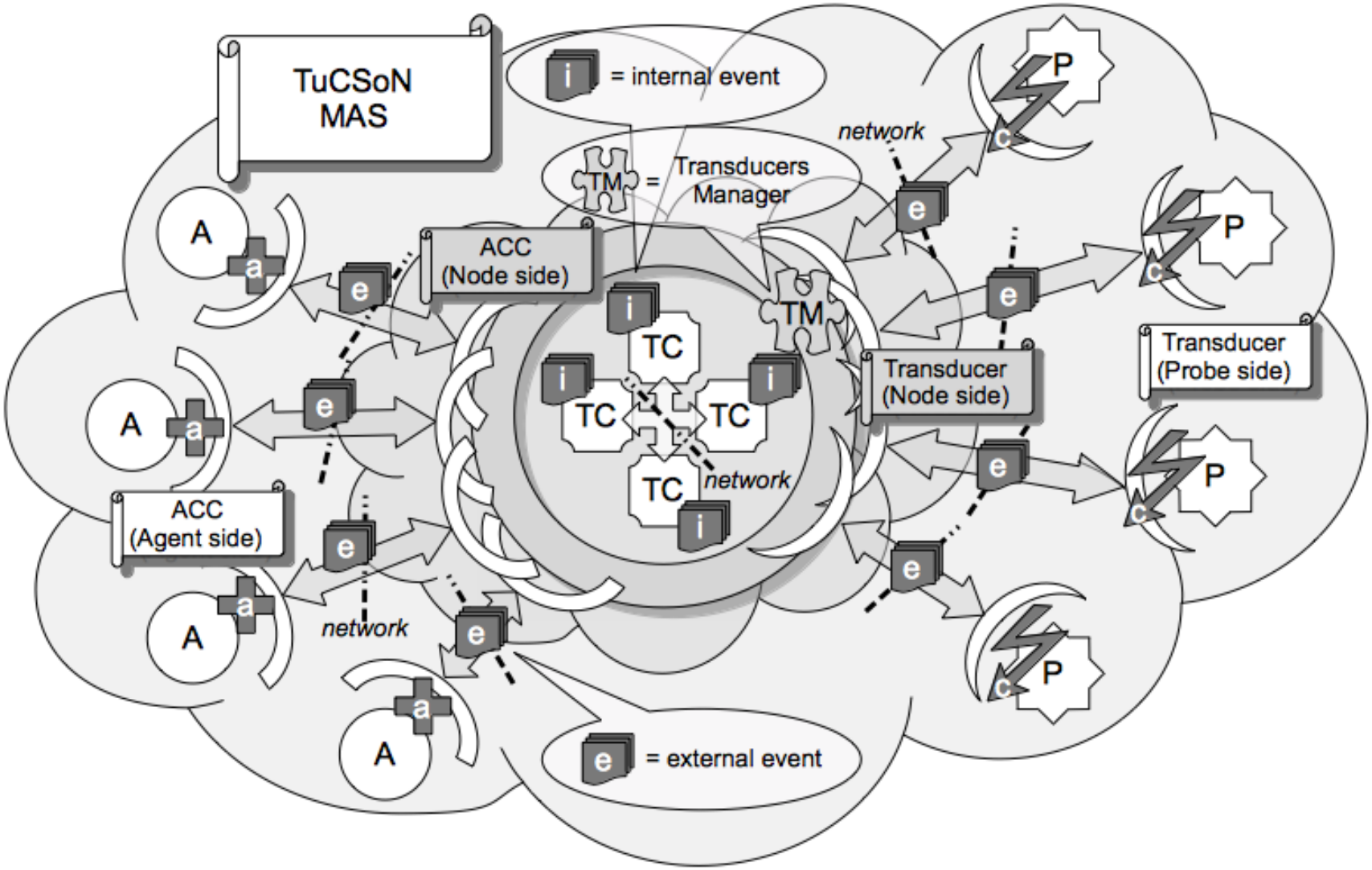
- Source and target of a tuple centre event can be any external resource
- A suitable *identification* scheme – both at the syntax and at the infrastructure level – is introduced for environmental resources
- The coordination language is extended to express explicit manipulation of environmental resources
- New *tuple centre predicates* are introduced, whose form is
  - $\langle EResId \rangle$ `?` `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

Dealing with Environment Change II

Transducers

- Specific environment events have to be translated into well-formed ReSpecT tuple centre events
- This is to be done at the infrastructure level, through a general-purpose schema that could be specialised according to the nature of any specific resource
- A *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.

TuCSoN Situated Architecture



An Example: TuCSoN Thermostat

- Package `alice.tucson.examples.situatedness` contains a simple example of how to exploit TuCSoN features for situated coordination
- A step-by-step *how-to* is reported in the TuCSoN main site at

`http://apice.unibo.it/xwiki/bin/download/TuCSoN/Documents/situatednesspdf.pdf`

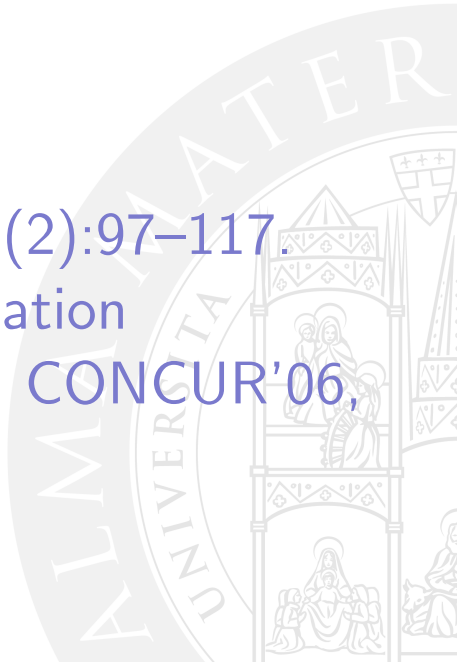
Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT
2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
3. Bibliography

Bibliography 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.

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.



Bibliography II

Omicini, A. and Mariani, S. (2013).
Coordination for situated MAS: Towards an event-driven architecture.
In Moldt, D. and Rölke, H., editors, *International Workshop on Petri Nets and Software Engineering (PNSE'13)*, volume 989 of *CEUR Workshop Proceedings*, pages 17–22. Sun SITE Central Europe, RWTH Aachen University.
Joint Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'13) and the International Workshop on Modeling and Business Environments (ModBE'13), co-located with the 34th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2013). Milano, Italy, 24–25 June 2013. Invited paper.

Bibliography III


Omicini, A., Ricci, A., and Viroli, M. (2005).
Time-aware coordination in ReSpecT.
In Jacquet, J.-M. and Picco, G. P., editors, *Coordination Models and Languages*, volume 3454 of *LNCS*, pages 268–282. Springer-Verlag. 7th International Conference (COORDINATION 2005), Namur, Belgium, 20–23 April 2005. Proceedings.

ReSpecT: **R**eaction **Spec**ification **T**uples
Advanced

Stefano Mariani
s.mariani@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015