ReSpecT: Reaction Specification Tuples Advanced

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

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM – Università di Bologna a Cesena

Academic Year 2015/2016
1. ReSpecT Java APIs
2. ReSpecT Advanced Constructs
3. Bibliography
These slides are adapted, arranged, integrated, starting from the official TuCSoN guide available at
1 ReSpecT Java APIs
   • Java Agents Using ReSpecT

2 ReSpecT Advanced Constructs
   • Timed-ReSpecT
   • ReSpecT-Events Observability
   • Situated Architecture
   • Space-Aware ReSpecT

3 Bibliography
Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT

2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
   - Space-Aware ReSpecT

3. Bibliography
Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT

2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
   - Space-Aware ReSpecT

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)".mvp);
   → 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
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
   - Space-Aware ReSpecT

3. Bibliography
Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT

2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
   - Space-Aware ReSpecT

3. Bibliography
Dining Philos in ReSpecT: How to Fix Starvation?

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.
ReSpecT Events Revised

By recalling [Omi07], we may note that a ReSpecT event is not limited to be any ReSpecT (either coordination or meta-coordination) primitive:

\[
\langle \text{SimpleTCEvent} \rangle ::= \langle \text{SimpleTCPredicate} \rangle \left( \langle \text{Tuple} \rangle \right) | \text{time} (\langle \text{Time} \rangle)
\]

1. A \text{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:

\[
\begin{align*}
\text{before}(\text{Time}) & \iff \text{the triggering event occurred before } \text{Time} \\
\text{after}(\text{Time}) & \iff \text{the triggering event occurred after } \text{Time} \\
\text{between}(\text{Time}, \text{Time'}) & \iff \text{after}(\text{Time}) \land \text{before}(\text{Time'})
\end{align*}
\]
Timed Dining Philosophers: Philosopher

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

With respect to Dining Philosopher’s protocol...

...this is left *unchanged* — and this is very convenient!
Timed Dining Philos: 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 =)

Let’s try!

Checkout example **TDiningPhilosophersTest** in package
alice.tucson.examples.timedDiningPhilos
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-awareness**, 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** [ORV05]

**Situated ReSpecT** [CO09]
Outline

1. ReSpecT Java APIs
   - Java Agents Using ReSpecT

2. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
   - Space-Aware ReSpecT

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:

\[
\langle ObservationPredicate \rangle ::= \langle EventView \rangle \_ \langle EventInformation \rangle
\]

\[
\langle EventView \rangle ::= \text{current} \mid \text{event} \mid \text{start}
\]

\[
\langle EventInformation \rangle ::= \text{predicate} \mid \text{tuple} \mid \text{source} \mid \text{target} \mid \text{time}
\]
Observability Semantics

Any combination of the following is admissible in ReSpecT, following formal grammar of $\langle ObservationPredicate \rangle$ in previous slide

$\langle EventView \rangle$ — 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

$\langle EventInformation \rangle$ — 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
   - Space-Aware ReSpecT

3. Bibliography
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.

⇒ ReSpecT tuple centres are able to capture general environment events, and to generally mediate process-environment interaction.
Thus, *situating* TuCSoN basically means making it capable of capturing environment events, and expressing general MAS-environment interactions [CO09, OM13].

⇒ 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:
  - `⟨EResId⟩ ? get(⟨Key⟩,⟨Value⟩)` enabling a tuple centre to get properties of environmental resources.
  - `⟨EResId⟩ ? set(⟨Key⟩,⟨Value⟩)` enabling a tuple centre to set properties of environmental resources.
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
Example & Further References

Let’s try

Check out example Thermostat in package alice.tucson.examples.situatedness

More on transducers & situatedness

- Papers
  - http://link.springer.com/chapter/10.1007/978-3-319-11692-1_9

- How-to
  - 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
   - Space-Aware ReSpecT

3. Bibliography
The availability of a plethora of mobile devices is pushing forward the needs for space-awareness of computations and systems. 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 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.
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 hardware 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.
The position of the coordination medium should be available to the \textit{coordination laws} it contains in order to make them capable of \textit{reasoning about space} — thus, to implement \textit{space-aware coordination laws}.

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

✓ a \textit{spatial event} should be \textit{generated} within a coordination medium, conceptually corresponding to \textit{changes in space}.

✓ then, such events should be \textit{captured} by the coordination medium, and used to \textit{activate} space-aware coordination laws.
Space-aware Middleware: TuCSoN on Android I
Space-aware Middleware: TuCSoN on Android II
Space-aware Middleware: TuCSoN on Android III
Space-aware Middleware: TuCSoN on Android IV

References

Codebase:
- https://bitbucket.org/smariani/tucsonandroid/overview
Outline

1 ReSpecT Java APIs
   - Java Agents Using ReSpecT

2 ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated Architecture
   - Space-Aware ReSpecT

3 Bibliography
Jacob Beal.  
A basis set of operators for space-time computations.  

Matteo Casadei and Andrea Omicini.  
Situated tuple centres in ReSpecT.  

ReSpecT: Reaction Specification Tuples Advanced

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

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

Academic Year 2015/2016