TuCSoN: Tuple Centres
Spread over the Network
ReSpecT Advanced

Marco Fabbri
(after Andrea Omicini, Stefano Mariani)
marco.fabbri28@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM—Università di Bologna

DS 2013/2014
Distributed Systems – 5th of May 2014
OUTLINE

1. ReSpecT Recap
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated ReSpecT

4. Bibliography
DISCLAIMER

These slides are adapted, rearranged, integrated starting from the official TuCSoN guide available at http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide by Andrea Omicini and Stefano Mariani.

Then, credits for all the stuff (text & images) goes to the original authors.

Credits for all the mistakes goes to the author.
OUTLINE

1. ReSpecT Recap
   • The ReSpecT Language
     • CLI Experiments I
     • The ReSpecT Virtual Machine
     • CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   • Timed-ReSpecT
   • ReSpecT-Events Observability
   • Situated ReSpecT

4. Bibliography
Focus on Guards

Currently, TuCSoN supports the following ReSpecT syntax*:

E (Event) — any TuCSoN primitive (except for get s and set s)

G (Guard) — any combination of

- **invocation**: true ↔ ReSpecT VM is in the invocation phase
- **completion**: true ↔ ReSpecT VM is in the completion phase
- **success**: true ↔ ReSpecT primitive succeeded
- **failure**: true ↔ ReSpecT primitive failed
- **endo**: true ↔ the cause of the event is the current tuple centre (tc)
- **exo**: true ↔ the cause of the event is not the current tc
- **intra**: true ↔ the target of the operation is the current tc
- **inter**: true ↔ the target of the operation is not the current tc
- **from_agent**: true ↔ the source of the ReSpecT event is an agent
- **from_tc**: true ↔ the source of the ReSpecT event is a tuple centre

R (Reaction) — any TuCSoN primitive (except for get s and set s) | any Prolog computation | any “mix” of the two

* For the full formal syntax, please refer to [Omicini, 2007], Table 1
OUTLINE

1. ReSpecT Recap
   - The ReSpecT Language
   - **CLI Experiments I**
     - The ReSpecT Virtual Machine
     - CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated ReSpecT

4. Bibliography
Meta-Coordination Experiments I

1. Launch a TuCSoN Node

   ```
   java -cp [...] [...].TucsonNodeService [-opts]
   ```

2. Launch the CLI

   ```
   java -cp [...] [...].CommandLineInterpreter [-opts]
   ```

3. Experiment with TuCSoN meta-coordination primitives
   - add a ReSpecT reaction to a tuple centre and try to trigger it
   - add two ReSpecT reactions and see how they “chain” together
   - play with `guards`
   - play with `distributed reactions`

4. Do whatever you want, ReSpecT has been formally proved to be Turing-complete [Denti et al., 1998] =)

---

* `tname @ netid : portno ? op_s`
OUTLINE

1. ReSpecT Recap
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
     - CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated ReSpecT

4. Bibliography
Focus on ReSpecT Main Execution Cycle

1. Whenever the invocation of a primitive by either an agent or a tuple centre is performed  
   → Invocation phase

2. When the tuple centre is idle (that is, no reaction is currently being executed)  
   → Triggering phase

3. Each reaction may trigger further reactions, again to be added to Re, and/or output events, representing link invocations  
   → Chaining & linking phase

4. Only when Re is finally empty  
   → Completion phase

This may give raise to further reactions, associated to the completion phase of the original invocation, and executed with the same semantics specified above for the invocation phase.
OUTLINE

1. ReSpecT Recap
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - **CLI Experiments II**

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated ReSpecT

4. Bibliography
1. Launch a TuCSoN Node
   java -cp [...] [...] .TucsonNodeService [-opts]

2. Launch the CLI
   java -cp [...] [...] .CommandLineInterpreter [-opts]

3. Try to detect & follow the ReSpecT VM main cycle by analyzing TuCSoN Node outputs on the console
   1. detect the invocation phase and intercept it with a ReSpecT reaction
   2. do the same with the completion phase
   3. detect the triggering phase and experiment success/failure of guard predicates evaluation
   4. detect the reaction execution phase and experiment their success/failure
   5. try to generate linking operations and detect them, possibly intercepting them in with a ReSpecT reaction
OUTLINE

1. ReSpecT Recap
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
   - Situated ReSpecT

4. Bibliography
External APIs

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

1. build a TucsonAgentId
   → TucsonAgentId aid = new TucsonAgentId("respect-programmer");

2. get a TuCSoN ACC allowing ReSpecT programming (e.g. SynchACC extends SpecificationSynchACC)
   → SynchACC acc = TucsonMetaACC.getContext(aid);

3. define the tuplecentre target of your meta-coordination operations
   → TucsonTupleCentreId tid = new TucsonTupleCentreId("default", "localhost", "20504");

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
   → if(op.isResultSuccess()){}

7. get requested operation result
   → LogicTuple res = op.getLogicTupleResult();
Extension APIs

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

2. extend alice.tucson.api.TucsonAgent base class

3. choose one of the given constructors, e.g.
   → protected RespectProgrammerAgent(String aid) throws TucsonInvalidAgentIdException { super(aid);
   }

4. override main() method with your agent business logic

5. get your ACC from the super-class
   → SpecificationSynchACC acc = getContext();

6. do what you want to do following steps 3 – 7 from previous slide

7. instantiate your agent and start its execution cycle (main()) by using method go()
   → new RespectProgrammerAgent(aid).go();
OUTLINE

1. ReSpecT Recap
   • The ReSpecT Language
   • CLI Experiments I
   • The ReSpecT Virtual Machine
   • CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   • Timed-ReSpecT
   • ReSpecT-Events Observability
   • Situated ReSpecT

4. 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:

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

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’) : true ⇔ predicate time(Time) which triggered the reaction has Time \geq Time’

after(Time’): true ⇔ predicate time(Time) which triggered the reaction has Time \leq Time’

between(Time,Time’) : true ⇔ after(Time) & before(Time’) is true
Time-Aware Coordination Medium

• Time-awareness is an essential feature to enable situatedness, that is the ability of a system to recognise 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 recognise 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 Recap
   • The ReSpecT Language
   • CLI Experiments I
   • The ReSpecT Virtual Machine
   • CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   • Timed-ReSpecT
     • ReSpecT-Events Observability
   • Situated ReSpecT

4. Bibliography
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:

\[
\langle \text{ObservationPredicate} \rangle \ ::= \langle \text{EventView} \rangle \langle \text{EventInformation} \rangle \\
\langle \text{EventView} \rangle \ ::= \text{current} | \text{event} | \text{start} \\
\langle \text{EventInformation} \rangle \ ::= \text{predicate} | \text{tuple} | \text{source} | \text{target} | \text{time}
\]
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 Recap
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   - Timed-ReSpecT
   - ReSpecT-Events Observability
     - Situated ReSpecT

4. Bibliography
Situated Coordination Medium

The enrichment of the ReSpecT language with the above described features enables the last extension made to the ReSpecT model, that is the already cited Situated ReSpecT. [Casadei and Omicini 2009]
Situation ReSpecT I

ReSpecT tuple centres for environment engineering

• Distributed systems are immersed into an environment, and should be reactive to events of any sort

• Also, coordination media should mediate any activity toward the environment, allowing for a fruitful interaction

⇒ ReSpecT tuple centres should be able to capture general environment events, and to generally mediate process-environment interaction
Situating ReSpecT II

Situating ReSpecT

• Thus, situating the ReSpecT language basically means making ReSpecT capable of capturing environment events, and expressing general MAS-environment interactions [Casadei and Omicini, 2009]

  ⇒ ReSpecT captures, reacts to, and observes general environment events

  ⇒ ReSpecT can explicitly interact with the environment
Coordination policies: talking about environment

ReSpecT is extended towards environment change by introducing some environment predicates and guards to get information about the environment status:

- observation
  - current_env(?Key,?Value),
  - start_env(?Key,?Value),
  - event_env(?Key,?Value),
- guard
  - from_env, to_env
Capturing general environment events

The ReSpecT event descriptor is extended, too

\[
\langle \text{Event} \rangle ::= \\
\langle \text{Predicate} \rangle \ (\langle \text{Tuple} \rangle ) \mid \text{time}(\langle \text{Time} \rangle ) \mid \\
\text{from}(\langle \text{Place} \rangle ) \mid \text{to}(\langle \text{Place} \rangle ) \mid \text{node}(\langle \text{Node} \rangle ) \mid \\
\text{env}(\langle \text{Key} \rangle ,\langle \text{Value} \rangle )
\]

making it possible to specify reactions to the occurrence of environment events

\[
\text{reaction(env(\langle ?Key, ?Value \rangle), Guard, Body).}
\]
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 ReSpecT language is extended to express explicit manipulation of environmental resources

• The body of a ReSpecT reaction can contain a tuple centre predicate of the form

  • \langle\text{EResId}\rangle ? \text{get}\langle\text{Key},\text{Value}\rangle
    enabling a tuple centre to get properties of environmental resources

  • \langle\text{EResId}\rangle ? \text{set}\langle\text{Key},\text{Value}\rangle
    enabling a tuple centre to set properties of environmental resources
Transducers

• Specific environment events have to be translated into well-formed ReSpecT tuple centre events

• This should be done at the infrastructure level, through a general-purpose schema that could be specialised according to the nature of any specific resource

• A ReSpecT 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.
Situatedness-related services in TuCSoN Middleware I

- the services for creation, (de)registration, etc. regarding transducers and probes have to be requested through well-formed tuples to be put into the special tuple centre called '$ENV$' (in truth, the same operations can be also requested directly through Java method call, but decoupling is lost and obviously this won't work in a distributed setting)
Situatedness-related services in TuCSoN Middleware II

- `createTransducerSensor(@Tcid, @Tclass, @Tid, @Pclass, @Pid)`
- `createTransducerActuator(@Tcid, @Tclass, @Tid, @Pclass, @Pid)`
- `addSensor(Class, @Pid, @Tid)`
- `addActuator(Class, @Pid, @Tid)`
- `removeResource(@Pid)`
- `changeTransducer(@Pid, @Tid)`
Implementing Transducers and Actuators

For Tranducers, subclass Java class `AbstractTransducer` implementing at least one method among `getEnv` and `setEnv` depending on whether the transducer is responsible for a sensor or an actuator (of course, both methods can be implemented if the transducer is responsible for a probe able to behave both as a sensor and as an actuator).

For Actuators, implementing interface `ISimpleProbe`. As for transducers, methods `readValue` and `writeValue` have to be implemented, respectively, for sensors and actuators (however, it is good practice to always implement both, raising exceptions or returning an error message in the non-exploited method). These methods should execute the “logic” of interaction with the specific probe, either software (e.g. in a simulation) or hardware (e.g. in a real-world deployment).
How TuCSoN agents can interact with these probes (or better, with the associated transducers)?

The last step is about programming suitable ReSpecT reactions within tuple centres deployed as mediators between agents and transducers. Tuple centres mediation is necessary because `getEnv` and `setEnv` are `ReSpecT-only primitives`, not accessible to TuCSoN agents.

This enforces decoupling and leverages encapsulation as well as separation of concerns.
OUTLINE

1. ReSpecT Recap
   • The ReSpecT Language
   • CLI Experiments I
   • The ReSpecT Virtual Machine
   • CLI Experiments II

2. ReSpecT Java APIs

3. ReSpecT Advanced Constructs
   • Timed-ReSpecT
   • ReSpecT-Events Observability
   • Situated ReSpecT

4. Bibliography
Bibliography I


TuCSoN: Tuple Centres Spread over the Network ReSpecT Advanced

Marco Fabbri
(after Andrea Omicini, Stefano Mariani)
marco.fabbri28@unibo.it

Dipartimento di Informatica: Scienza e Ingegneria (DISI)
ALMA MATER STUDIORUM—Università di Bologna

DS 2013/2014
Distributed Systems – 5th of June 2014