ReSpecT: Reaction Specification Tuples Basics

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. Programming TuCSoN Tuple Centres
2. Programming Tuple Centres in ReSpecT
3. Bibliography
These slides are adapted, arranged, integrated, starting from the official TuCSoN guide available at

Programming TuCSoN Tuple Centres
- Meta-Coordination Language
- Meta-Coordination Primitives

Programming Tuple Centres in ReSpecT
- The ReSpecT Language
- CLI Experiments I
- The ReSpecT Virtual Machine
- CLI Experiments II

Bibliography
Outline

1 Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2 Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3 Bibliography
Outline

1. Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2. Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3. Bibliography
Meta-Coordination Language

- The TuCSoN meta-coordination language allows agents to program ReSpeCt tuple centres by executing *meta-coordination operations*. 
- TuCSoN provides coordinables with *meta-coordination primitives*, allowing agents to read, write, consume ReSpeCt specification tuples in tuple centres and also to synchronise on them.
- Meta-coordination operations are built out of meta-coordination primitives and of the ReSpeCt *specification languages*:
  - the *specification language*
  - the *specification template language*

In the following, whenever unspecified, we assume that

\[
\text{reaction}(E,G,R) \text{ belongs to the specification language, and } \\
\text{reaction}(ET,GT,RT) \text{ belongs to the specification template language.}
\]
Given that the TuCSoN coordination medium is the logic-based ReSpecT tuple centre, both the specification and the specification template languages are logic-based too—and defined by ReSpecT.

More precisely
- any ReSpecT reaction is an admissible TuCSoN specification tuple...
- ... and an admissible TuCSoN specification template...
Meta-Coordination Operations

- Any TuCSoN *meta-coordination operation* is invoked by a source agent on a target tuple centre, which is in charge of its execution.

- In the same way as TuCSoN coordination operations, any meta-coordination operation have two phases:
  - **invocation** — the *request* from the source agent to the target tuple centre, carrying all the information about the invocation.
  - **completion** — the *response* from the target tuple centre back to the source agent, including all the information about the operation execution by the tuple centre.
Abstract Syntax

- The abstract syntax of a meta-coordination operation $op_s$ invoked on a target tuple centre $tcid$ is

  $$tcid \ ? \ op_s$$

  where $tcid$ is the tuple centre full name

- Given the structure of the full name of a tuple centre, the general abstract syntax of a TuCSoN meta-coordination operation is

  $$tname \ @ \ netid \ : \ portno \ ? \ op_s$$
Outline

1. Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2. Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3. Bibliography
Meta-Coordination Primitives

- The TuCSoN meta-coordination language provides the following meta-coordination primitives to build meta-coordination operations:
  - `out_s`
  - `rd_s`, `rdp_s`
  - `in_s`, `inp_s`
  - `no_s`, `nop_s`
  - `get_s`
  - `set_s`

- As you can see, meta-primitives perfectly match basic primitives, thus allowing a uniform access to both the tuple space and the specification space in a TuCSoN tuple centre.
Outline

1. Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2. Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3. Bibliography
Outline

1. Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2. Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3. Bibliography
As a *behaviour specification language*, ReSpecT

- enables the definition of *computations* within a tuple centre—called *reactions*
- makes it possible to associate such reactions to *events* occurring in a tuple centre

**ReSpecT twofold interpretation**

So, ReSpecT has both a *declarative* and a *procedural* part
As a specification language, ReSpecT allows *events* to be declaratively associated to *reactions* by means of specific logic tuples, called *specification tuples*, whose form is $\text{reaction}(E, G, R)$ [Omicini, 2007].

**Specification tuples definition**

In short, given a ReSpecT event $Ev$, a specification tuple $\text{reaction}(E, G, R)$ associates a reaction $R\theta$ to $Ev$ if and only if $\theta = \text{mgu}(E, Ev)^a$ and guard predicate $G$ is true.

---

$^a\text{mgu}$ is the most general unifier, as defined in logic programming.
ReSpecT Syntax

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

---

1. Upcoming TuCSoN release is TuCSoN-1.12.0.0301.
2. For the full formal syntax, please refer to [Omicini, 2007], Table 1.
By adopting the declarative interpretation, a ReSpecT tuple centre has then a *twofold nature*: a *theory of communication* – the set of the ordinary tuples – and a *theory of coordination*—the set of the specification tuples.

**Intelligency**

This allows in principle intelligent agents to *reason* about the state of *collaboration activities* and to possibly affect their dynamics.
Outline

1. Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2. Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3. Bibliography
Meta-Coordination Experiments I

1. Launch a TuCSoN Node
   
   \texttt{java \ -cp \ [...] \ [...]}.TucsonNodeService \ [-opts]

2. Launch the CLI
   
   \texttt{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 \textit{distributed reactions}

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

1 Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2 Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3 Bibliography
Procedural ReSpecT

As a reaction language, ReSpecT enables *reactions* to be procedurally defined in terms of sequences of *logic reaction goals*, each one either succeeding or failing.

**Reactions semantics**

- A ReSpecT reaction *as a whole* succeeds if and only if all its reaction goals succeed, and fails otherwise.
- Each reaction is executed *sequentially* with a *transactional* semantics: hence, a failed reaction has *no effect* on the state of a ReSpecT tuple centre.
- All the reactions triggered by a ReSpecT event are executed *before* serving any other event: thus, agents are *transparent* to any reactions chain and perceive them as *atomic*—along with the (meta-)coordination primitive invoked.
Whenever the invocation of a primitive by either an agent or a tuple centre is performed

Invocation

1. an (admissible) ReSpecT event is generated and...
2. ...reaches its (the primitive) target tuple centre
3. where it is orderly inserted in a sort of input queue (InQ)
ReSpecT Main Execution Cycle II

When the tuple centre is idle (that is, no reaction is currently being executed)

**Triggering**

1. the first event $\epsilon$ in $InQ$ (according to a FIFO policy) is moved to the multiset $Op$ of the requests to be served
2. consequently, reactions to the invocation phase of $\epsilon$ are triggered by adding them to the multiset $Re$ of the triggered reactions waiting to be executed
3. all triggered reactions in $Re$ are then executed in a *non-deterministic order*—sequential, transactional semantics
Each reaction may trigger

- further reactions, again to be added to Re
- output events representing link invocations: such events are
  1. added to the multiset $Out$ of the outgoing events
  2. then moved to the tuple-centre outgoing queue $OutQ$ at the end of the reaction execution—if and only if successful
ReSpecT Main Execution Cycle IV

Completion

Only when $Re$ is finally empty:

1. requests waiting to be served in $Op$ are possibly executed by the tuple centre
2. operation/link completions are sent back to invokers

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 Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2 Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3 Bibliography
Meta-Coordination Experiments II

1. Launch a TuCSoN Node
   ```java
   java -cp [...] [...]\.TucsonNodeService [-opts]
   ```

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

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

1. Programming TuCSoN Tuple Centres
   - Meta-Coordination Language
   - Meta-Coordination Primitives

2. Programming Tuple Centres in ReSpecT
   - The ReSpecT Language
   - CLI Experiments I
   - The ReSpecT Virtual Machine
   - CLI Experiments II

3. Bibliography
On the expressive power of a language for programming coordination media.
In *1998 ACM Symposium on Applied Computing (SAC’98)*, pages 169–177, Atlanta, GA, USA. ACM.
Special Track on Coordination Models, Languages and Applications.

5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA’06), CONCUR’06, Bonn, Germany, 31 August 2006. Post-proceedings.
ReSpecT: Reaction Specification Tuples Basics

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