TuCSoN: Tuple Centres
Spread over the Network
ReSpecT Basics

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 – 29th of May 2014
OUTLINE

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

2. ReSpecT
   • The ReSpecT Language
   • CLI Experiments I
   • The ReSpecT Virtual Machine
   • CLI Experiments II

3. 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. Programming Tuple Centres
   • Meta-Coordination Language
   • Meta-Coordination Primitives

2. 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 reaction(E,G,R) belongs to the specification language, and reaction(ET,GT,RT) belongs to the specification template language.
Specification & Specification Template Languages

• 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 \texttt{op\_s} invoked on a target tuple centre \texttt{tcid} is

\[
\texttt{tcid} \ ? \ \texttt{op\_s}
\]

(\textit{where tcid is the tuple centre full name})

• Given the structure of the full name of a tuple centre, the \textbf{general abstract syntax} of a TuCSoN meta-coordination operation is

\[
\texttt{tname} @ \texttt{netid} : \texttt{portno} \ ? \ \texttt{op\_s}
\]
OUTLINE

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

2. 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 9 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.
Basic Meta-Primitives

outs(E,G,R)
writes a specification tuple reaction(E,G,R) in the target tuple centre; after the operation is successfully executed, the specification tuple reaction(E,G,R) is returned as a completion.

rd_s(ET,GT,RT)
looks for a specification tuple reaction(E,G,R) matching reaction(ET,GT,RT) in the target tuple centre; if a matching specification tuple is found when the operation is served, the execution succeeds by returning reaction(E,G,R); otherwise, the execution is suspended to be resumed and successfully completed when a matching specification tuple will be finally found in and returned from the target tuple centre.

in_s(ET,GT,RT)
looks for a specification tuple reaction(E,G,R) matching reaction(ET,GT,RT) in the target tuple centre; if a matching specification tuple is found when the operation is served, the execution succeeds by removing and returning reaction(E,G,R); otherwise the execution is suspended to be resumed and successfully completed when a matching specification tuple will be finally found in, removed and returned from the target tuple centre.
Predicative Meta-Primitives

\( \text{rdp}_s(ET, GT, RT) \)

predicative (non-suspensive) version of \( \text{rd}_s(ET, GT, RT) \);
if a matching specification tuple is not found, the execution **fails** (operation outcome is **FAILURE**) and
\( \text{reaction}(ET, GT, RT) \) is returned

\( \text{inp}_s(ET, GT, RT) \)

predicative version of \( \text{in}_s(ET, GT, RT) \); if a matching specification tuple is not found, the execution **fails**, no specification tuple is removed from the target tuple centre and \( \text{reaction}(ET, GT, RT) \) is returned
Test-for-Absence Meta-Primitives

\texttt{nos(ET,GT,RT)}
looks for a specification tuple \texttt{reaction(E,G,R)} matching \texttt{reaction(ET,GT,RT)} in the target tuple centre; if no matching specification tuple is found when the operation is served, the execution succeeds by returning \texttt{reaction(ET,GT,RT)}; otherwise, the execution is suspended to be resumed and successfully completed when no matching specification tuples can any longer be found in the target tuple centre.

\texttt{nop_s(ET,GT,RT)}
predicative version of \texttt{no_s(ET,GT,RT)}; if a matching specification tuple is found the execution \texttt{fails} and \texttt{reaction(E,G,R)} is returned.
Space Meta-Primitives

get_s
reads all the specification tuples in the target tuple centre and returns them as a list; if no specification tuple occurs in the target tuple centre at execution time, the empty list is returned and the execution succeeds anyway

set_s([(E1,G1,R1),..., (En,Gn,Rn)])
overwrites the target tuple centre with the list of specification tuples reaction(E1,G1,R1), ..., reaction(En,Gn,Rn); when the execution is completed, the list of specification tuples is successfully returned
1. Programming Tuple Centres
   • Meta-Coordination Language
   • Meta-Coordination Primitives

2. ReSpecT
   • The ReSpecT Language
   • CLI Experiments I
   • The ReSpecT Virtual Machine
   • CLI Experiments II

3. Bibliography
ReSpecT In Brief

As a behaviour specification language, ReSpecT (Reaction Spec Tuples):

• 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.
Declarative ReSpecT

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 \( E_v \), a specification tuple \( \text{reaction}(E,G,R) \) associates a reaction \( R\theta \) to \( E_v \) if and only if \( \theta = \text{mgu}(E,E_v)* \) and *guard predicate* \( G \) is true.

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

**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
ReSpecT Tuple Centres Knowledge

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

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

3. Bibliography
Meta-Coordinating Experiments I

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

2. Launch the CLI
   ```java
   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
1. Programming Tuple Centres
   • Meta-Coordination Language
   • Meta-Coordination Primitives

2. 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.
ReSpecT Main Execution
Cycle I

Whenever the *invocation* of a primitive by either an agent or a tuple centre is performed

**Invocation**

- an (admissible) ReSpecT event is generated and…
- …reaches its (the primitive) target tuple centre
- 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**

- the first event $\varepsilon$ in InQ (according to a FIFO policy) is moved to the multiset $Op$ of the requests to be served

- consequently, reactions to the invocation phase of $\varepsilon$ are triggered by adding them to the multiset $Re$ of the triggered reactions waiting to be executed

- all triggered reactions in $Re$ are then executed in a *non-deterministic order*—sequential, transactional semantics
Each reaction may trigger

**Chaining & linking**

- 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

Only when \textit{Re} is finally empty

\textbf{Completion}

- requests waiting to be served in \textit{Op} are possibly executed by the tuple centre

- operation/link completions are sent back to invokers

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

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

2. 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 -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

TuCSoN: Tuple Centres Spread over the Network ReSpecT Basics

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 – 29th of May 2014