

ReSpecT: Reaction **S**pecification **T**uples Basics

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



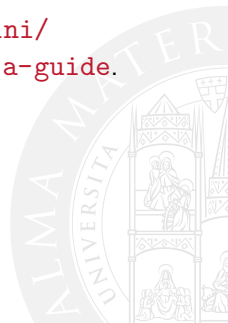
- 1 Programming TuCSoN Tuple Centres
- 2 Programming Tuple Centres in ReSpecT
- 3 Bibliography



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](http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide).



- 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
 - Dining Philosophers: First Attempt

- 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
 - Dining Philosophers: First Attempt
- 3 Bibliography



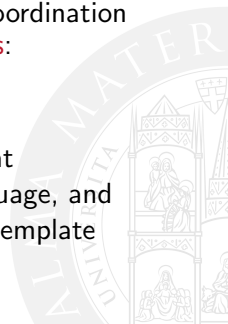
Outline

- 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
 - Dining Philosophers: First Attempt
- 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 $\text{reaction}(E, G, R)$ belongs to the specification language, and $\text{reaction}(ET, GT, RT)$ belongs to the specification template language



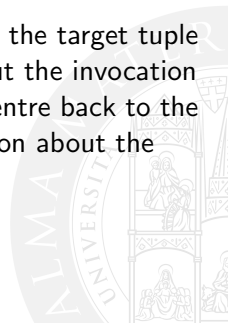
Specification & Specification Template Languages

- 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 *op_s* invoked on a target tuple centre *tcid* is

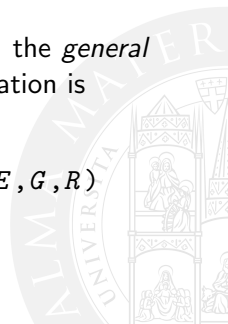
tcid ? op_s

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

default @ localhost : 20504 ? out_s(*E,G,R*)



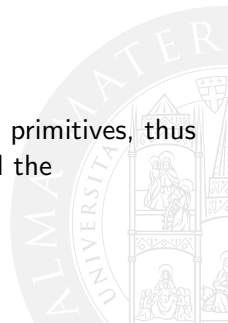
Outline

- 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
 - Dining Philosophers: First Attempt
- 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
 - Dining Philosophers: First Attempt
- 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
 - Dining Philosophers: First Attempt
- 3 Bibliography



Reaction Specification Tuples (ReSpecT)

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



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 `reaction( $E, G, R$ )` [Omi07].

Specification tuples definition

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

^a mgu is the most general unifier, as defined in logic programming.



ReSpecT Syntax

Currently¹, TuCSoN supports the following ReSpecT syntax²:

E (Event) — any TuCSoN primitive (except for `get_s` and `set_s`)

G (Guard) — any combination of

`invocation` : true $\iff$ ReSpecT VM is in the **invocation** phase

`completion` : true $\iff$ ReSpecT VM is in the **completion** phase

`success` : true $\iff$ ReSpecT primitive succeeded

`failure` : true $\iff$ ReSpecT primitive failed

`endo` : true $\iff$ the **cause** of the event is the current tuple centre (tc)

`exo` : true $\iff$ the *cause* of the event *is not* the current tc

`intra` : true $\iff$ the target of the operation is the current tc

`inter` : true $\iff$ the target of the operation *is not* the current tc

`from_agent` : true $\iff$ the **source** of the ReSpecT event is an agent

`from_tc` : true $\iff$ 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

¹Upcoming TuCSoN release is TuCSoN-1.12.0.0301.

²For the full formal syntax, please refer to [Omi07], 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



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
 - Dining Philosophers: First Attempt
- 3 Bibliography



Meta-Coordination Experiments (1/2)

Let's try!

- ➊ Launch a TuCSoN node on default port
- ➋ Launch the CLI
- ➌ Launch TuCSoN **Inspector** tool — that is, class **InspectorGUI** in package `alice.tucson.introspection.tools`
- ➍ 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*
- ➎ Launch class **BagOfTaskTest** in package `alice.tucson.examples.respect.bagOfTask`
- ➏ Do whatever you want, ReSpecT has been formally proved to be Turing-complete [DNO98] =)

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
 - Dining Philosophers: First Attempt
- 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 ReSpecTevent 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

- 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 ϵ 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 ϵ 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



ReSpecT Main Execution Cycle III

Chaining & linking

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



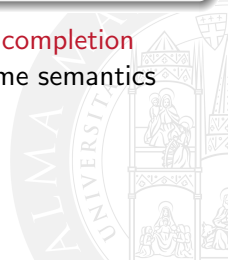
ReSpecT Main Execution Cycle IV

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 rise 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
 - Dining Philosophers: First Attempt
- 3 Bibliography



Meta-Coordination Experiments (2/2)

Let's try!

- ❶ Launch a TuCSoN Node, Inspector, and CLI
- ❷ 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
 - Dining Philosophers: First Attempt
- 3 Bibliography



Usage Example: Dining Philos I

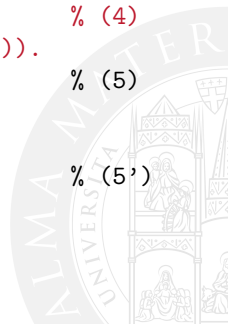
```
philosopher(I,J) :-  
    think,                                % thinking  
    table ? in(chops(I,J)),              % waiting to eat  
    eat,                                  % eating  
    table ? out(chops(I,J)),             % waiting to think  
    !, philosopher(I,J).
```

Results

- + fairness, no deadlock
- + trivial philosopher's interaction protocol
- ? shared resources handled properly?
- ? starvation still possible?

Usage Example: Dining Philos II

```
reaction( out(chops(C1,C2)), (operation, completion), ( % (1)
    in(chops(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)) )).
```



Usage Example: Dining Philos III

Results

protocol no deadlock

protocol fairness

protocol trivial philosopher's interaction protocol

tuple centre shared resources handled properly

: (starvation still possible

Keep starvation in mind...

...later on we will fix the issue, using ReSpecT :)



ReSpecT API

Full ReSpecT API

Class `Respect2PLibrary` in package `alice.respect.api` implements all predicates available within ReSpecT reactions, including observation predicates, and also guards

Let's try!

Check out examples in package `alice.tucson.examples.*`

- 1 `.respect.bagOfTask`
- 2 `.diningPhilos`
- 3 `.distributedDiningPhilos` — linking primitives here



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
 - Dining Philosophers: First Attempt
- 3 Bibliography



Bibliography



Enrico Denti, Antonio Natali, and Andrea Omicini.

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, 27 February – 1 March 1998. ACM. Special Track on Coordination Models, Languages and Applications.

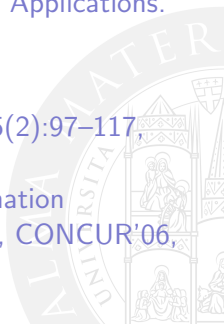


Andrea Omicini.

Formal ReSpecT in the A&A perspective.

Electronic Notes in Theoretical Computer Science, 175(2):97–117, June 2007.

5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 August 2006. Post-proceedings.



ReSpecT: Reaction **S**pecification **T**uples Basics

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

