TuCSoN: Tuple Centres Spread over the Network

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 – 16th of May 2014
DISCLAIMER

These slides are an excerpt 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.
Part I

Basic TuCSoN
Outline

1. Basic Model & Language
2. Basic Architecture
3. Basic Technology
Part 1: Basic TuCSoN

1. Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2. Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3. Basic Technology
   - Middleware
   - Tools
TuCSoN (Tuple Centres Spread over the Network) is a model for the coordination of distributed processes, as well as of autonomous, intelligent & mobile agents [Omicini and Zambonelli, 1999]

Main URLs

**URL** http://tucson.unibo.it/

**FaceBook** http://www.facebook.com/TuCSoNCoordinationTechnology

**Bitbucket** http://bitbucket.org/smariani/tucson/

**SourceForge** http://sf.net/projects/tucson/
Basic Entities

- **TuCSoN agents** are the *coordinables*
- **ReSpecT tuple centres** are the (default) *coordination media* [Omicini and Denti, 2001]
- **TuCSoN nodes** represent the basic *topological abstraction*, which host the tuple centres
- agents, tuple centres, and nodes have *unique identities* within a TuCSoN system

System

Roughly speaking, a TuCSoN system is a collection of agents and tuple centres working together in a possibly-distributed set of nodes
Basic Interaction

- since agents are *pro-active* entities, and tuple centres are *reactive* entities, coordinables need *coordination operations* in order to *act* over coordination media: such operations are built out of the TuCSoN coordination language
- agents interact by exchanging tuples through tuple centres using TuCSoN coordination primitives, altogether defining the coordination language
- tuple centres provide the shared space for tuple-based communication (*tuple space*), along with the programmable behaviour space for tuple-based coordination (*specification space*)

System

Roughly speaking, a TuCSoN system is a collection of agents and tuple centres interacting in a possibly-distributed set of nodes
Basic Topology

- agents and tuple centres are spread over the network
- tuple centres belong to nodes
- agents live anywhere on the network, and can interact with the tuple centres hosted by any reachable TuCSoN node
- agents could in principle move independently of the device where they run, tuple centres are permanently associated to one device

System

Roughly speaking, a TuCSoN system is a collection of possibly-distributed nodes and agents interacting with the nodes’ tuple centres.
Basic Mobility

- agents could in principle *move independently* of the device where they run [Omicini and Zambonelli, 1998]
- tuple centres are essentially associated to one device, possibly *mobile*—so, tuple centre mobility is dependent on their hosting device

**System**

Roughly speaking, a TuCSoN system is a collection of possibly-distributed nodes, associated to possibly-mobile devices agents, interacting with the nodes’ tuple centres.
Part 1: Basic TuCSoN

1. Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2. Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3. Basic Technology
   - Middleware
   - Tools
Nodes

- each node within a TuCSoN system is univocally identified by the pair $< \text{NetworkId}, \text{PortNo} >$, where
  - $\text{NetworkId}$ is either the IP number or the DNS entry of the device hosting the node,
  - $\text{PortNo}$ is the port number where the TuCSoN coordination service listens to the invocations for the execution of coordination operations.
- correspondingly, the abstract syntax for the identifier of a TuCSoN node hosted by a networked device netid on port portno is $\text{netid : portno}$
an admissible name for a tuple centre is any first-order ground logic term

since each node contain at most one tuple centre for each admissible name, each tuple centre is uniquely identified by its admissible name associated to the node identifier

the TuCSoN full name of a tuple centre tname on a node netid : portno is

\[ tname @ \text{netid} : \text{portno} \]

the full name of a tuple centre works as a tuple centre identifier in a TuCSoN system
Agents

• an admissible name for an agent is any Prolog first-order ground logic term [Lloyd, 1984]
• when it enters a TuCSoN system, an agent assigned a universally unique identifier (UUID)\(^1\)
• if an agent aname is assigned UUID uuid, its full name is aname : uuid

\(^1\)http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html
Part 1: Basic TuCSoN

1. Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2. Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3. Basic Technology
   - Middleware
   - Tools
the **TuCSoN coordination language** allows agents to interact with tuple centres by executing *coordination operations*

TuCSoN provides coordinables with *coordination primitives*, allowing agents to read, write, consume tuples in tuple spaces, and to synchronise on them

*coordination operations* are built out of *coordination primitives* and of the *communication languages*:

- the *tuple language*
- the *tuple template language*

In the following, whenever unspecified, we assume that *Tuple* belongs to the tuple language, and *TupleTemplate* belongs to the tuple template language
both the tuple and the tuple template languages depend on the sort of the tuple centres adopted by TuCSoN

given that the default TuCSoN coordination medium is the logic-based ReSpecT tuple centre, both the tuple and the tuple template languages are logic-based, too

more precisely

- any Prolog atom is an admissible TuCSoN tuple
- any Prolog atom is an admissible TuCSoN tuple template

as a result, the default TuCSoN tuple and tuple template languages coincide
Coordination Operations

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

- Any TuCSoN operation has 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.
the abstract syntax of a coordination operation \( \text{op} \) invoked on a target tuple centre \( \text{tcid} \) is

\[
\text{tcid} \ ? \ \text{op}
\]

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

given the structure of the full name of a tuple centre, the *general abstract syntax* of a TuCSoN coordination operation is

\[
\text{tname} @ \text{netid} : \text{portno} ? \ \text{op}
\]
Part 1: Basic TuCSoN

1. Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2. Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3. Basic Technology
   - Middleware
   - Tools
The TuCSoN coordination language provides the following 9 \textit{coordination primitives} to build coordination operations:

- out, rd, in
- rdp, inp
- no, nop
- get, set
**out(Tuple)** writes *Tuple* in the target tuple space; after the operation is successfully executed, *Tuple* is returned as a completion.

**rd(TupleTemplate)** looks for a tuple matching *TupleTemplate* in the target tuple space; if a matching *Tuple* is found when the operation is first served, the execution succeeds by returning *Tuple*; otherwise, the execution is suspended, to be resumed and successfully completed when a matching *Tuple* is finally found on the target tuple space, and returned.

**in(TupleTemplate)** looks for a tuple matching *TupleTemplate* in the target tuple space; if a matching *Tuple* is found when the operation is first served, the execution succeeds by removing and returning *Tuple*; otherwise, the execution is suspended, to be resumed and successfully completed when a matching *Tuple* is finally found on the target tuple space, removed, and returned.
Predicative Operations

\texttt{rdp(TupleTemplate)} looks for a tuple matching \textit{TupleTemplate} in the target tuple space; if a matching \textit{Tuple} is found when the operation is served, the execution succeeds, and \textit{Tuple} is returned; otherwise the execution fails, and \textit{TupleTemplate} is returned;

\texttt{inp(TupleTemplate)} looks for a tuple matching \textit{TupleTemplate} in the target tuple space; if a matching \textit{Tuple} is found when the operation is served, the execution succeeds, \textit{Tuple} is removed from the target tuple space, and returned; otherwise the execution fails, no tuple is removed from the target tuple space, and \textit{TupleTemplate} is returned;
no(TupleTemplate) looks for a Tuple matching TupleTemplate in the target tuple space; if no matching tuple is found in the target tuple space when the operation is first served, the execution succeeds, and TupleTemplate is returned; otherwise, the execution is suspended, to be resumed and successfully completed when no matching tuples can any longer be found in the target tuple space, then TupleTemplate is returned.

nop(TupleTemplate) looks for a Tuple matching TupleTemplate in the target tuple space; if no matching tuple is found in the target tuple space when the operation is served, the execution succeeds, and TupleTemplate is returned; otherwise, if a matching Tuple is found, the execution fails, and Tuple is returned.
Space Operations

get reads all the *Tuples* in the target tuple space, and returns them as a list; if no tuple occurs in the target tuple space at execution time, the empty list is returned, and the execution succeeds anyway

set(*Tuples*) rewrites the target tuple spaces with the list of *Tuples*; when the execution is completed, the list of *Tuples* is successfully returned
Part 1: Basic TuCSoN

1 Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2 Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3 Basic Technology
   - Middleware
   - Tools
a TuCSoN system is first of all a characterised by the (possibly
distributed) collection of TuCSoN nodes hosting a TuCSoN service

• a node is characterised by the networked device hosting the service,
  and by the network port where the TuCSoN service listens to
  incoming requests

Multiple nodes on a single device

Many TuCSoN nodes can in principle run on the same networked device,
each one listening on a different port
The default port number of TuCSoN is 20504

- so, an agent can invoke operations of the form
  \[ \text{tname} @ \text{netid} ? \text{op} \]
  without specifying the node port number \text{portno}, meaning that the agent intends to invoke operation \text{op} on the tuple centre \text{tname} of the default node \text{netid} = 20504 hosted by the networked device \text{netid}.

- any other port could in principle be used for a TuCSoN node.

- the fact that a TuCSoN node is available on a networked device does \textit{not} imply that a node is also available on the same unit on the default port—so the default node is \textit{not} ensured to exist, generally speaking.
given an admissible tuple centre name \( t_{\text{name}} \), tuple centre \( t_{\text{name}} \) is an admissible tuple centre

the \textit{coordination space} of a TuCSoN node is defined as the collection of all the admissible tuple centres

any TuCSoN node provides agents with a \textit{complete} coordination space, so that in principle any coordination operation can be invoked on any admissible tuple centre belonging to any TuCSoN node
Default Tuple Centre

- every TuCSoN node defines a **default tuple centre**, which responds to any operation invocation received by the node that do not specify the target tuple centre

---

**Default tuple centre**

- The *default tuple centre* of any TuCSoN node is named **default**

- as a result, agents can invoke operations of the form

  ```
  @ netid : portno ? op
  ```

  without specifying the tuple centre name `tname`, meaning that they intend to invoke operation `op` on the default tuple centre of the node `netid : portno` hosted by the networked device `netid`
combining the notions of default tuple centre and default port, agents can also invoke operations of the form

@ netid ? op

meaning that they intend to invoke operation op on the default tuple centre of the default node netid : 20504 hosted by the networked device netid
Part 1: Basic TuCSoN

1. Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2. Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3. Basic Technology
   - Middleware
   - Tools
Global coordination space

- the TuCSoN **global coordination space** is defined at any time by the collection of all the tuple centres available on the network, hosted by a node, and identified by their full name

- a TuCSoN agent running on any networked device has at any time the whole TuCSoN global coordination space available for its coordination operations through invocations of the form:

  ```
  tname @ netid : portno ? op
  ```

  which invokes operation `op` on the tuple centre `tname` provided by node `netid : portno`
Local Coordination Space

- given a networked device $\text{netid}$ hosting one or more TuCSoN nodes, the TuCSoN local coordination space is defined at any time by the collection of all the tuple centres made available by all the TuCSoN nodes hosted by $\text{netid}$
- an agent running on the same device $\text{netid}$ that hosts a TuCSoN node can exploit the *local coordination space* to invoke operations of the form

$$\text{tname : portno ? op}$$

which invokes operation $\text{op}$ on the tuple centre $\text{tname}$ locally provided by node $\text{netid : portno}$
by exploiting the notions of default node and default tuple centre, the following invocations are also admissible for any TuCSoN agent running on a device netid:

- \( : \text{portno} \ ? \ \text{op} \)
  invoking operation \( \text{op} \) on the default tuple centre of node
  \( \text{netid} : \text{portno} \)
- \( \text{tname} \ ? \ \text{op} \)
  invoking operation \( \text{op} \) on the \( \text{tname} \) tuple centre of default node
  \( \text{netid} : 20504 \)
- \( \text{op} \)
  invoking operation \( \text{op} \) on the default tuple centre of default node
  \( \text{netid} : 20504 \)
Part 1: Basic TuCSoN

1. Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Operations

2. Basic Architecture
   - Nodes & Tuple Centres
   - Coordination Spaces

3. Basic Technology
   - Middleware
   - Tools
Technology Requirements

- TuCSoN is a **Java-based** middleware
- TuCSoN is also **Prolog-based**: it is based on the tuProlog Java-based technology for
  - first-order logic tuples
  - primitive & identifier parsing
  - ReSpecT specification language & virtual machine
Java & Prolog Agents

TuCSoN middleware provides

- **Java API** for extending Java programs with TuCSoN coordination primitives
  - package `alice.tucson.api.*`
- **Java classes** for programming TuCSoN agents in Java
  - `alice.tucson.api.TucsonAgent` provides a ready-to-use thread, whose main can directly use TuCSoN coordination primitives
- **Prolog libraries** for extending tuProlog programs with TuCSoN coordination primitives
  - `alice.tucson.api.Tucson2PLibrary` provides tuProlog agents with the ability to use TuCSoN primitives
  - by including the `:-load_library(path/to/Tucson2PLibrary)` directive in its Prolog theory
Java APIs I

Package `alice.tucson.api`

Most APIs are made available through package `alice.tucson.api`.

**TucsonAgentId** — exposes methods to get a TuCSoN agent ID, and to access its fields. Required to obtain an ACC.

- `getAgentId()`: `Object` — to get the full agent ID
- `getAgentName()`: `String` — to get only the agent name

**TucsonMetaACC** — provides TuCSoN agents with an ACC. The ACC is mandatory to interact with a TuCSoN tuple centre.

- `getContext(TucsonAgentId, String, int)`: `EnhancedACC` — to get an ACC from the (specified) TuCSoN node
**TucsonTupleCentreId** — exposes methods to get a TuCSoN tuple centre ID, and to access its fields. Required to perform TuCSoN operations on the ACC.

- **getName()**: `String` — to get the tuple centre local name
- **getNode()**: `String` — to get the tuple centre host’s IP number
- **getPort()**: `int` — to get the tuple centre host’s listening port number

**ITucsonOperation** — exposes methods to access the result of a TuCSoN operation.

- **isResultSuccess()**: `boolean` — to check operation success
- **getLogicTupleResult()**: `LogicTuple` — to get operation result
- **getLogicTupleListResult()**: `List<LogicTuple>` — to get operation result— to be used with bulk primitives and get...
Java APIs III

**TucsonAgent** — base abstract class for user-defined TuCSoN agents. Automatically builds the TucsonAgentId and gets the EnhancedACC.

- **main()**: `void` — to be overridden by business logic of the user-defined agent
- **getContext()**: EnhancedACC — to get ACC for the user-defined agent
- **go()**: `void` — to start main execution of the user-defined agent

**SpawnActivity** — base abstract class for user-defined TuCSoN *activities* to be spawned by a spawn operation. Provides a simplified syntax for TuCSoN operation invocations.

- **doActivity()**: `void` — to override with your spawned activity business logic
- **out(LogicTuple)**: LogicTuple — out TuCSoN operation
- **unop(LogicTuple)**: LogicTuple — unop TuCSoN operation
Java APIs IV

**TuCSoN2PLibrary** — allows tuProlog agents to access the TuCSoN platform by exposing methods to manage ACCs, and to invoke TuCSoN operations.

- `acquire_acc_1(Struct): boolean` — to get an ACC for your tuProlog agent
- `out_2(Term, Term): boolean` — out TuCSoN operation
- `unop_2(Term, Term): boolean` — unop TuCSoN operation

Furthermore...

Package `alice.tucson.api` obviously contains also all the ACCs provided by the TuCSoN platform—among which `EnhancedACC`. Please refer to Slides 90–96 for the complete list, and to Slide 97 for an overview.
Other APIs are made available through package Alice.logictuple. In particular, those required to manage TuCSoN tuples.

**LogicTuple** — exposes methods to build a TuCSoN tuple/template and to get its arguments.

- `parse(String)` — to encode a given string into a TuCSoN tuple/template
- `getName()` — to get the functor name of the tuple
- `getArg(int)` — to get the tuple argument at given position
TupleArgument — represents TuCSoN tuples arguments (tuProlog terms), thus provides the means to access them.

- `parse(String)`: TupleArgument — to encode the given string into a tuProlog tuple argument
- `getArg(int)`: TupleArgument — to get the tuple argument at given position
- `isVar()`: boolean — to test if the tuple argument is a tuProlog Var (other similar methods provided)
- `intValue()`: int — to get the int value of the tuple argument (other similar methods provided)
APIs to programatically boot & kill a TuCSoN service are provided by class \texttt{TucsonNodeService} in package \texttt{alice.tucson.service}.

- constructors to init the TuCSoN service (possibly on a given port)
- methods to install & shutdown the TuCSoN service
  - \texttt{install()}: void
  - \texttt{shutdown()}: void
- entry point to launch a TuCSoN node from the command line

\footnotetext[2]{Always an EnhancedACC in current implementation TuCSoN-1.10.6.0208}
TuCSoN adopts logic tuples as its main communication language however, Java tuples can also be used for straightforward communication among TuCSoN Java agents

Java *tuples* and *templates* can be used
- A Java tuple is an ordered collection of Java *tuple values*
- A tuple value has one of the following *tuple types*: double, float, int, literal, long
- A Java template is an ordered collection of Java *tuple values* and *tuple variables*
- A tuple variable has either a tuple type or special type any
HelloWorld example

In alice.tucson.examples.helloWorld.JTuples, a TuCSoN agent

- builds a TuCSoN Agent Identifier, gets an ACC, and defines the TuCSoN Tuple Centre Identifier of the target tuple centre
  
  ```java
  TucsonAgentId aid = null;
  SynchACC acc = TucsonMetaACC.getContext(aid);
  final TucsonTupleCentreId tid =
      new TucsonTupleCentreId("default", "localhost", "20504");
  ```

- builds the tuple to write, and outputs it in the tuple in the tuple centre
  ```java
  final IJTuple tuple = new JTuple(new JVal("hello"));
  tuple.addArg(new JVal("world"));
  ITucsonOperation op = acc.out(tid, tuple, null);
  ```

- checks the tuple in the tuple centre, by building the proper template and reading the tuple in the tuple centre, and finally releases the ACC
  ```java
  final IJTupleTemplate template = new JTupleTemplate(new JVal("hello"));
  template.addArg(new JVar(JArgType.LITERAL));
  op = acc.rdp(tid, template, null); if (op.isResultSuccess()) { ... }
  acc.exit();
  ```
The TuCSoN Basic Technology

Java Tuples III

Main packages: tuples

IJTuple in alice.tuples.javatuples.api — interface representing Java tuples

JTuple in alice.tuples.javatuples.impl — class implementing Java tuples. Java tuples are implemented as ordered collections of IJVal objects

IJVal in alice.tuples.javatuples.api — interface representing Java tuples values (just values, not variables)

JVal in alice.tuples.javatuples.impl — class implementing Java tuples values. Java tuples values can be of one of the tuple types defined by enumeration JArgType. Methods to convert Java tuples values into primitive Java types are provided.

JArgType in alice.tuples.javatuples.api — enumeration defining the admissible Java tuple types: double, float, int, literal, long. Special type any is reserved for usage in Java tuple templates.
Java Tuples IV

Main packages: templates

**IJTupleTemplate** in alice.tuples.javatuples.api — interface representing Java tuple templates

**JTupleTemplate** in alice.tuples.javatuples.impl — class implementing Java tuple templates. Java tuple templates are implemented as ordered collections of **IJArg** objects.

**IJArg** in alice.tuples.javatuples.api — interface representing both Java tuples values and Java tuple variables for templates

**IJVar** in alice.tuples.javatuples.api — interface representing Java tuple template variables

**JVar** in alice.tuples.javatuples.impl — class implementing Java tuple template variables. Java tuple templates variables can be of any tuple type declared in **JArgType** enumeration, including **any**. Matching of variables against templates is typed. Variables of type any match any value.
Java Tuples V

Main packages: operations

getJTupleResult method of interface ITucsonOperation can be used to retrieve the result of an operation. It returns an object of class Tuple, which should then be queried if it is a tuple or a template (usual operator instanceof will do the job) so as to be managed accordingly.
given any networked device running a Java VM, a TuCSoN node can be booted to make it provide a TuCSoN service

a TuCSoN service can be started through the alice.tucson.service Java API, e.g.

java -cp TuCSoN-1.10.6.0208.jar alice.tucson.service.TucsonNodeService -portno 20506

the node service is in charge of
- listening to incoming operation invocations on the associated port of the device
- dispatching them to the target tuple centres
- returning the operation completions
a TuCSoN node service provides the complete coordination space

tuple centres in a node are either *actual* or *potential*: at any time in a given node

- actual tuple centres are admissible tuple centres that already *do* have a reification as a run-time abstraction
- potential tuple centres are admissible tuple centres that *do not* have a reification as a run-time abstraction, yet

the node service is in charge of making *potential* tuple centres *actual* as soon as the first operation on them is received and served.
Part 1: Basic TuCSoN

1 Basic Model & Language
   • Basic Model
   • Naming
   • Basic Language
   • Basic Operations

2 Basic Architecture
   • Nodes & Tuple Centres
   • Coordination Spaces

3 Basic Technology
   • Middleware
   • Tools
Command Line Interface (CLI)

- Shell interface for human agents / programmers, e.g.
  
  ```
  java -cp TuCSoN-1.10.6.0208.jar alice.tucson.service.tools.CommandLineInterpreter 
  -netid localhost -portno 20505 -aid myCLI
  ```
## CLI Syntax

\[
\langle TucsonOp \rangle ::= \langle TcName \rangle @ \langle IpAddress \rangle : \langle PortNo \rangle ? \langle Op \rangle \\
\langle TcName \rangle ::= \text{Prolog ground term} \\
\langle IpAddress \rangle ::= \text{localhost | IP address} \\
\langle PortNo \rangle ::= \text{port number} \\
\langle Op \rangle ::= \text{out(T) | in(TT) | rd(TT) | no(TT) | inp(T) | rdp(TT) | nop(TT) | get() | set([T1,...,Tn]) | out_all(TT,TL) | in_all(TT,TL) | rd_all(TT,TL) | no_all(TT,TL) | uin(TT) | urd(TT) | uno(TT) | uinp(TT) | urdp(TT) | unop(TT) | out_s(E,G,R) | in_s(ET,GT,RT) | rd_s(ET,GT,RT) | no_s(ET,GT,RT) | inp_s(ET,GT,RT) | rdp_s(ET,GT,RT) | nop_s(ET,GT,RT) | get_s() | set_s([(E1,G1,R1),...,,(En,Gn,Rn)])}
\]

\[
T,T1,...,Tn ::= \text{tuple (Prolog term)} \\
TT ::= \text{tuple template (Prolog term)} \\
TL ::= \text{list of tuples (Prolog list of terms)} \\
E,E1,...,En ::= \text{ReSpecT event} \\
G,G1,...,Gn ::= \text{ReSpecT guard predicate} \\
R,R1,...,Rn ::= \text{ReSpecT reaction body} \\
ET ::= \text{ReSpecT event template} \\
GT ::= \text{ReSpecT guard template} \\
RT ::= \text{ReSpecT reaction body template}
\]
TuCSoN Inspector

A GUI tool to monitor the TuCSoN coordination space & ReSpecT VM—to some extent, actually it’s still in development

- to launch the **Inspector** tool
  
  ```
  java -cp TuCSoN-1.10.6.0208.jar alice.tucson.introspection.tools.Inspector
  ```

- available options are
  
  - **-aid** — the name of the Inspector Agent
  - **-netid** — the IP address of the device hosting the TuCSoN Node to be inspected...
  - **-portno** — ...its listening port...
  - **-tcname** — ...and the name of the tuplecentre to monitor
Using the Inspector Tool I

- if you launched it without specifying the full name of the target tuplecentre to inspect, choose it from the GUI
if you launched it giving the full name of the target tuplecentre to inspect, choose what to inspect inside that tuplecentre
TuCSoN Inspector IV

What to inspect

In the *Sets* tab you can choose whether to inspect

Tuple Space — the *ordinary* tuples space state

Specification Space — the (ReSpecT) *specification* tuples space state

Pending Ops — the *pending* TuCSoN operations set, that is the set of the currently suspended issued operations (waiting for completion)

ReSpecT Reactions — the *triggered* (ReSpecT) reactions set, that is the set of specification tuples (recursively) triggered by the issued TuCSoN operations
TuCSoN Inspector V

Tuple Space view

In the *Tuple Space* view you can

- **proactively** observe the space state, thus getting any change of state, or **reactively** observe it, that is getting updates only when requested—through the *Observe!* button in the *Observation* tab

- **filter** displayed tuples according to a given admissible Prolog template—through the *Match!* button in the *Filter* tab

- **dump** (filtered) observations on a given log file—in the *Log* tab
TuCSoN Inspector VI

```
# observations: 8

temperature:room:1,1),celsius:15)
temperature:room:1,2),celsius:17)
temperature:room:2,1),celsius:16)
temperature:room:2,2),celsius:14)
humidity:room:1,1),percentage:30)
humidity:room:1,2),percentage:27)
humidity:room:2,1),percentage:22)
humidity:room:2,2),percentage:35)
```

Output:
- dump observations on file: inspector-tuples.log

Template:
- Filter observed tuples using the following template:

Ready for tuples inspection.
In the *Specification Space* view you can

- load a ReSpecT specification from a file...
- **edit & set** it to the current tuplecentre—through the `<set_s>` button
- get the ReSpecT specification from the current tuplecentre—through the `<get_s>` button...
- ...save it to a given file (or to the default one named `default.rsp`)—button Save (or Save As)
TuCSoN Inspector VIII

```plaintext
reaction
  out{precious|X},
  ',|completion,success},
  out|backup|X})
).

reaction
  in|X},
  ',|completion,success},
  out|consumed|X})
).
```

Specification read. line 11
In the *Pending Ops* view you can

- **proactively** observe pending TuCSoN operations, thus getting any new update whenever available, or **reactively** observe it, that is getting updates only when requested—through the *Observe!* button in the *Observation* tab

- **filter** displayed TuCSoN operations according to a given admissible Prolog template—through the *Match!* button in the *Filter* tab

- **dump** (filtered) observations on a given log file—in the *Log* tab

\(^a\text{filtering is based on operation tuples solely a.t.m.}\)
TuCSoN Inspector X

Pending TuCSoN operations set of tuplecentre <default@localhost:20504>

# observations: 2

in:operation:req:1,who:cli01,res:R}) from <cli01> to '<@':default,'@':localhost,20504)>
in:operation:req:1,who:cli02,res:R}) from <cli02> to '<@':default,'@':localhost,20504)>

filtering

- Filter observed tuples using the following template:

Ready for pending operations inspection.
In the ReSpecT Reactions view you are notified upon any ReSpecT reaction triggered in the observed tuplecentre and can dump such notifications on a given log file.
TuCSoN Inspector XII

Triggered ReSpecT reaction set of tuplecentre < default@localhost:20504 >
reaction < reaction(out:precious,pin:1234)),out:backup,pin:1234)))) > SUCCEEDED @ 9:20:44.
reaction < reaction(in:precious,pin:1234)),out:consumed,precious,pin:1234)))) > SUCCEEDED @ 9:20:52.

store

Ready for ReSpecT reactions triggering notification.
Part 2: Advanced TuCSoN

4 Advanced Model
- Bulk Primitives
  - Coordinative Computation
  - Uniform Primitives
  - Organisation
  - Agent Coordination Contexts

5 Advanced Architecture
- Node Architecture
- Situated Architecture

6 Programming Tuple Centres
- Meta-Coordination Language
- Meta-Coordination Operations
Bulk Primitives: The Idea

- bulk coordination primitives are required in order to obtain significant efficiency gains for a large class of coordination problems involving the management of more than one tuple with a single coordination operation [Rowstron, 1996]
- instead of returning one single matching tuple, bulk operations return list of matching tuples
- in case of no matching tuples, they successfully return an empty list of tuples: so, bulk primitives always succeed
Bulk Primitives: Simple Examples

For instance, let us assume that the default tuple centre contains just 3 tuples: 2 colour(white) and 1 colour(black)

- the invocation of a \texttt{rd\_all(color(X))} succeeds and returns a list of 3 tuples, containing 2 colour(white) and 1 colour(black) tuples
- the invocation of a \texttt{rd\_all(color(black))} succeeds and returns a list of 1 tuples, containing 1 colour(black) tuples
- the invocation of a \texttt{no\_all(color(black))} succeeds and returns an empty list of tuples
- the invocation of a \texttt{no\_all(color(blue))} succeeds and returns a list of 2 tuples, containing 2 colour(white) tuples
- the invocation of a \texttt{no\_all(color(blue))} succeeds and returns a list of 3 tuples, containing 2 colour(white) and 1 colour(black) tuples

On the other hand, \texttt{out\_all(Tuples)} just takes a list of \texttt{Tuples} and simply put them all in the target tuple space.
The TuCSoN coordination language provides the following 4 *bulk coordination primitives* to build coordination operations:

- `out_all`
- `rd_all`
- `in_all`
- `no_all`


Space-aware coordination in ReSpecT.
In Baldoni, M., Baroglio, C., Bergenti, F., and Garro, A., editors, From Objects to Agents,
volume 1099 of CEUR Workshop Proceedings, pages 1–7, Turin, Italy. Sun SITE Central
Europe, RWTH Aachen University.
XIV Workshop (WOA 2013). Workshop Notes.

Semantic tuple centres.
Science of Computer Programming.
Special Issue on Self-Organizing Coordination.

On the semantics of tuple-based coordination models.
In 1999 ACM Symposium on Applied Computing (SAC’99), pages 175–182, New York,
NY, USA. ACM.
Special Track on Coordination Models, Languages and Applications.

Towards a notion of agent coordination context.
In Marinescu, D. C. and Lee, C., editors, Process Coordination and Ubiquitous Computing,


An algebraic approach for modelling organisation, roles and contexts in MAS.
Special Issue: Process Algebras and Multi-Agent Systems.

RBAC for organisation and security in an agent coordination infrastructure.

Time-aware coordination in ReSpecT.

Coordination of mobile information agents in TuCSoN.
*Internet Research*, 8(5):400–413.

TuCSoN: Tuple Centres
Spread over the Network

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 – 16th of May 2014