TuCSoN: Tuple Centres Spread over the Network 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
These slides are adapted, arranged, integrated, starting from the official TuCSoN guide available at

1 Basic Model & Language
   • Basic Model
   • Naming
   • Basic Language
   • Basic Primitives

2 Basic Architecture
   • Nodes & Tuple Centres

3 Basic Technology
   • Middleware
   • Tools
   • CLI Experiments

4 Basic Java APIs
   • Java Apps & Java TuCSoN Agents
   • Java Experiments

5 Bibliography
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
TuCSoN is a model for the coordination of distributed processes, as well as of autonomous agents [Omicini and Zambonelli, 1999]

References

Main Page http://tucson.unibo.it/
FaceBook http://www.facebook.com/TuCSoNCoordinationTechnology
Bitbucket http://bitbucket.org/smariani/tucson/
Basic Entities

- **TuCSoN agents** are the *coordinables*
- **ReSpecT tuple centres** are the *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 view**

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 whereas tuple centres are (mostly) *reactive*, the coordinables need *coordination operations* in order to act over the coordination media.

- Such operations are built out of the TuCSoN coordination language, defined by the collection of TuCSoN coordination primitives that agents can use to interact—by exchanging tuples.

- 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 view**

Roughly speaking, a TuCSoN system is a collection of agents and tuple centres *coordinating* in a (possibly) distributed set of nodes.
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
Nodes

- Each node within a TuCSoN system is *univocally identified* by the pair \(< \text{NetworkId}, \text{PortNo} >\), where
  - *NetworkId* is the IP number of the device hosting the node
  - *PortNo* is the port number where the TuCSoN coordination service listens incoming requests

- Correspondingly, the abstract syntax\(^1\) of TuCSoN nodes identifiers hosted by a networked device `netid` on port `portno` is
  \[
  \text{netid : portno}
  \]

---

\(^1\)Actually, this is also the concrete syntax used by TuCSoN to parse nodes ID
An *admissible name* for a tuple centre is *any* Prolog-like, first-order logic *ground term*\(^2\) [Lloyd, 1984]

Each tuple centre is uniquely identified by its admissible name associated to the node identifier

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

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

\(^2\) *Ground* roughly means containing no variables
Agents

- An admissible name for an agent is any Prolog-like, first-order logic ground term too.
- When it enters a TuCSoN system, an agent is assigned a universally unique identifier (UUID)\(^3\).
- If an agent `aname` is assigned UUID `uuid`, its full name is `aname : uuid`.

---

\(^3\) [http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html](http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html)
Outline

1 Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
     - Basic Primitives

2 Basic Architecture
   - Nodes & Tuple Centres

3 Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4 Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5 Bibliography
Coordination Language

- 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.
- 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.
Given that the 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 first-order logic Prolog atom is an admissible TuCSoN tuple...
- ...and an admissible TuCSoN tuple template
Coordination Operations

- Any 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.
Abstract Syntax

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

$$tcid \ ? \ op$$

where $tcid$ is the tuple centre *full name*

- Given the structure of the full name of a tuple centre, the *general abstract syntax*\(^4\) of a TuCSoN coordination operation is

$$tname \ @ \ netid : \ portno \ ? \ op$$

\(^4\)Actually, this is also the concrete syntax used by TuCSoN to parse coordination operations, even inside ReSpecT reactions
Outline

1 Basic Model & Language
   • Basic Model
   • Naming
   • Basic Language
   • Basic Primitives

2 Basic Architecture
   • Nodes & Tuple Centres

3 Basic Technology
   • Middleware
   • Tools
   • CLI Experiments

4 Basic Java APIs
   • Java Apps & Java TuCSoN Agents
   • Java Experiments

5 Bibliography
The TuCSoN coordination language provides the following 9 basic\textsuperscript{5} coordination primitives to build coordination operations:

- out
- rd, rdp
- in, inp
- no, nop
- get
- set

\textsuperscript{5}We will see others in next lesson.
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
A **TuCSoN 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 run on the same networked device, as long as each one is 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}^{6}
- Any other port can be used for a TuCSoN node listening service (we will see how to change it in a few slides)

\[^{6}\text{Meaning that the agent intends to invoke operation op on the tuple centre tname of the default node netid : 20504, hosted by the networked device netid}\]
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`.
By combining the notions of default node and default tuple centre, the following invocations are also admissible for any TuCSoN agent running on a device \texttt{netid}:

- \texttt{: portno ? op}
  invoking operation \texttt{op} on the default tuple centre of node
  \texttt{netid : portno}

- \texttt{tname ? op}
  invoking operation \texttt{op} on the \texttt{tname} tuple centre of default node
  \texttt{netid : 20504}

- \texttt{op}
  invoking operation \texttt{op} on the default tuple centre of default node
  \texttt{netid : 20504}
The TuCSoN Basic Technology

Outline

1 Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Primitives

2 Basic Architecture
   - Nodes & Tuple Centres

3 Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4 Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5 Bibliography
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
Technology Requirements

- TuCSoN is a **Java-based** middleware (Java 7 is enough)
- TuCSoN is also **Prolog-based**: it is based on the tuProlog Java-based technology for
  - first-order logic tuples
  - primitives & identifiers parsing
  - ReSpecT specification language & virtual machine

\[7\] Last digits in TuCSoN version number (TuCSoN-1.12.0.0301) are for the tuProlog version, hence tuProlog version 3.0.1 now
TuCSoN middleware provides

- **Java API** for using TuCSoN coordination services from Java programs
  - package `alice.tucson.api.*`
- **Prolog API** for using TuCSoN coordination services from tuProlog programs
  - `alice.tucson.api.Tucson2PLibrary` enables tuProlog agents to use TuCSoN primitives
  - use directive `:-load_library(path/to/Tucson2PLibrary)` to load the library
Given any networked device running a Java VM, a TuCSoN node can be started to provide TuCSoN coordination services.

```java
java -cp tucson.jar:2p.jar alice.tucson.service.TucsonNodeService
   -port 20505
```

The node service is in charge of:
- listening to incoming operation invocations
- dispatching them to the target tuple centre
- returning the operations completion to the source agent
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
     - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
Command Line Interpreter (CLI) I

- Shell interface for humans

```java
java -cp tucson.jar:2p.jar
   alice.tucson.service.tools.CommandLineInterpreter
   -netid localhost -port 20505 -aid myCLI
```

```
panzutoidota:tucson ste$ java -cp TuCSoN-1.10.2.0205.jar alice.tucson.service.tools.CommandLineInterpreter -netid localhost -port 20
505 -aid myCLI
[CommandLineInterpreter]: Booting TuCSoN Command Line Interpreter...
[CommandLineInterpreter]: Version TuCSoN-1.10.2.0205
[CommandLineInterpreter]:
[CommandLineInterpreter]: Demanding for TuCSoN default ACC on port < 20505 >
[CommandLineInterpreter]: Spawning CLI TuCSoN agent...
[CommandLineInterpreter]:
[CLI]: CLI agent listening to user...
[CLI]: ?> help
[CLI]: TuCSoN CLI Syntax:
[CLI]:
[CLI]: tcName@ipAddress:port ? CMD
[CLI]:
[CLI]: where CMD can be:
[CLI]:
[CLI]: out(Tuple)
[CLI]: in(TupleTemplate)
[CLI]: rd(TupleTemplate)
[CLI]: no(TupleTemplate)
[CLI]: inp(TupleTemplate)
```
Command Line Interpreter (CLI) II

CLI Syntax

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

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

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
TuCSoN Experiments I

1. Launch a local TuCSoN Node
   ```
   java -cp tucson.jar:2p.jar alice.tucson.service.TucsonNodeService
   ```

2. Launch the CLI on that node
   ```
   java -cp tucson.jar:2p.jar alice.tucson.service.tools.CommandLineInterpreter
   ```

3. Experiment with the semantics of basic TuCSoN primitives
   - rd vs. in
   - rd/in vs. rdp/inp
   - rd vs. no

4. Experiment with LINDA-like coordination by working with multiple CLIs

5. Experiment with TuCSoN distribution by working with multiple nodes (and possibly multiple CLIs)
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
# Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
External APIs

To enable a Java application to use the TuCSoN technology, do the following:

1. build a TucsonAgentId to be identified by the TuCSoN system
2. get a TuCSoN ACC to enable interaction with the TuCSoN system
3. define the tuplecentre target of your coordination operations
4. build a tuple using the communication language
5. perform the coordination operation using a coordination primitive
6. check requested operation success
7. get requested operation result

See example HelloWorld in package alice.tucson.examples.helloWorld within TuCSoN distribution
Extension APIs

To create a TuCSoN agent, do the following:

1. extend `alice.tucson.api.TucsonAgent` base class
2. choose one of the given constructors
3. override the `main()` method with your agent business logic
4. get your ACC from the super-class
5. do what you want to do following steps 3 – 7 from previous slide
6. instantiate your agent and start its execution cycle (`main()`) by using method `go()`

See example `HelloWorldAgent` in package `alice.tucson.examples.helloWorld` within TuCSoN distribution
Outline

1 Basic Model & Language
   - Basic Model
   - Naming
   - Basic Language
   - Basic Primitives

2 Basic Architecture
   - Nodes & Tuple Centres

3 Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4 Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5 Bibliography
TuCSoN Experiments II

1. Launch a local TuCSoN Node
   
   ```java
   java -cp tucson.jar:2p.jar alice.tucson.service.TucsonNodeService
   ```

2. Launch the **HelloWorld** Java program within TuCSoN distribution

3. Launch the **HelloWorldAgent** TuCSoN agent within TuCSoN distribution

4. If you feel confident enough, experiment with packages
   
   - `ds.lab.tucson.messagePassing`
   - `ds.lab.tucson.rpc`
   - `ds.lab.tucson.masterWorkers`

   and, read the comments
Outline

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

2. Basic Architecture
   - Nodes & Tuple Centres

3. Basic Technology
   - Middleware
   - Tools
   - CLI Experiments

4. Basic Java APIs
   - Java Apps & Java TuCSoN Agents
   - Java Experiments

5. Bibliography
Bibliography


TuCSoN: Tuple Centres Spread over the Network 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