TuCSoN: Tuple Centres Spread over the Network 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. Basic Model & Language

2. Basic Architecture

3. Basic Technology

4. Basic Java APIs

5. Bibliography
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 [OZ99]

References

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

- TuCSoN agents are the *coordinables*
- ReSpecT tuple centres are the *coordination media* [OD01]
- 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
  \[
  \text{netid} : \text{portno} \\
  (\text{localhost} : 20504)
  \]

\(^1\) Actually, this is also the concrete syntax used by TuCSoN to parse nodes ID
Tuple Centres

- An admissible name for a tuple centre is any Prolog-like, first-order logic ground term\(^2\) [Llo84]
- 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

\[
\text{tname @ netid : portno}
\]
\[
\text{(default @ localhost : 20504)}
\]

\(^2\) *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 \textit{aname} is assigned UUID \textit{uuid}, its full name is

\[ \textit{aname} : \textit{uuid} \]

(stefano : 4baad505-ad2f-4ac4-b30b-bc3705a2c87a)

\(^3\)

http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html

Mariani & Omicini (DISI, Univ. Bologna)
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 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.
Tuple & Tuple Template Languages

- 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 \( \text{op} \) 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\(^4\) of a TuCSoN coordination operation is:
  \[
  \text{tname} \ @ \ \text{netid} : \text{portno} \ ? \ \text{op}
  \]
  (default @ localhost : 20504 ? out(t(hi)))

---

\(^4\)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 coordination primitives to build coordination operations:

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

\(^5\)We will see others in next lesson
The TuCSoN Basic Architecture

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
Default Node

Default port

The default port number of TuCSoN is 20504

- So, an agent can invoke operations of the form
  \[ tname @ netid ? op \]
  without specifying the node port number \texttt{portno} \footnote{\textsuperscript{6}}

- Any other port can be used for a TuCSoN node listening service (we will see how to change it in a few slides)

\footnote{\textsuperscript{6}}\textsuperscript{6}Meaning that the agent intends to invoke operation \texttt{op} on the tuple centre \texttt{tname} of the default node \texttt{netid} : 20504, hosted by the networked device \texttt{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 `netid`:

- `portno ? op`
  invoKing operation `op` on the default tuple centre of node `netid : portno`

- `tname ? op`
  invoKing operation `op` on the `tname` tuple centre of default node `netid : 20504`

- `op`
  invoKing operation `op` on the default tuple centre of default node `netid : 20504`
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
Java & Prolog Agents

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
Service

- Given any networked device running a Java VM, a TuCSoN node can be started to provide TuCSoN coordination services
  
  \[ \text{java -cp tucon.jar:2p.jar alice.tucson.service.TucsonNodeService} \]
  \[ \text{-port 20505} \]

- 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

Let’s try!

1. Open a console, position yourself into the folder where tucon and 2p jars are, then type the command above — on Windows, replace “:” with “;”

2. Try to launch another TuCSoN node on a different portno
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 20505 -aid myCLI
[CommandLineInterpreter]: ______________________________________________________________________________________
[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]: ______________________________________________________________________________________
[CLI]: TuCSoN CLI Syntax:
[CLI]: [CLI]: tcName@ipAddress:port ? CMD
[CLI]: [CLI]: where CMD can be:
[CLI]: [CLI]: out(Tuple)
[CLI]: [CLI]: in(TupleTemplate)
[CLI]: [CLI]: rd(TupleTemplate)
[CLI]: [CLI]: no(TupleTemplate)
[CLI]: [CLI]: inp(TupleTemplate)
```
**CLI Syntax**

\[
\langle TucsonOp \rangle ::= \langle TcName \rangle @ \langle IpAddress \rangle : \langle PortNo \rangle ? \langle Op \rangle \\
\langle TcName \rangle ::= \text{Prolog ground term} \\
\langle IpAddress \rangle ::= \text{localhost} | \text{IP address} \\
\langle PortNo \rangle ::= \text{port number} \\
\langle Op \rangle ::= \text{out}(T) | \text{in}(TT) | \text{rd}(TT) | \text{no}(TT) | \text{inp}(TT) | \text{rdp}(TT) | \text{nop}(TT) | \text{get}() | \text{set}([T_1, \ldots, T_n]) | \\
\text{out\_all}(TL) | \text{in\_all}(TT, TL) | \text{rd\_all}(TT, TL) | \text{no\_all}(TT, TL) | \\
\text{uin}(TT) | \text{urd}(TT) | \text{uno}(TT) | \text{uinp}(TT) | \text{urdp}(TT) | \text{unop}(TT) | \\
\text{out\_s}(E, G, R) | \text{in\_s}(ET, GT, RT) | \text{rd\_s}(ET, GT, RT) | \text{no\_s}(ET, GT, RT) | \\
\text{inp\_s}(ET, GT, RT) | \text{rdp\_s}(ET, GT, RT) | \text{nop\_s}(ET, GT, RT) | \\
\text{get\_s}() | \text{set\_s}([(E_1, G_1, R_1), \ldots, (E_n, G_n, R_n)])
\]

T, T_1, \ldots, T_n ::= \text{tuple (Prolog term)} \\
TT ::= \text{tuple template (Prolog term)} \\
TL ::= \text{list of tuples (Prolog list of terms)} \\
E, E_1, \ldots, E_n ::= \text{ReSpecT event} \\
G, G_1, \ldots, G_n ::= \text{ReSpecT guard predicate} \\
R, R_1, \ldots, R_n ::= \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

Let’s try!

Launch Java class HelloWorld in package alice.tucson.examples.helloWorld within TuCSoN distribution and check out code comments
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()`

Let’s try!
Launch Java class `HelloWorldAgent` in package `alice.tucson.examples.helloWorld` within TuCSoN distribution and check out code 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
TuCSoN Experiments II

Package `alice.tucson.examples.*`

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

2. `.helloWorld` package

3. `.messagePassing` package

4. `.rpc` package

5. `.masterWorkers` package
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
John W. Lloyd.  
*Foundations of Logic Programming.*  

Andrea Omicini and Enrico Denti.  
From tuple spaces to tuple centres.  

Andrea Omicini and Franco Zambonelli.  
Coordination for Internet application development.  
Special Issue: Coordination Mechanisms for Web Agents.
TuCSoN: Tuple Centres Spread over the Network 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