TuCSoN: **Tu**ple **C**entres **S**pread **o**ver the **N**etwork Basics

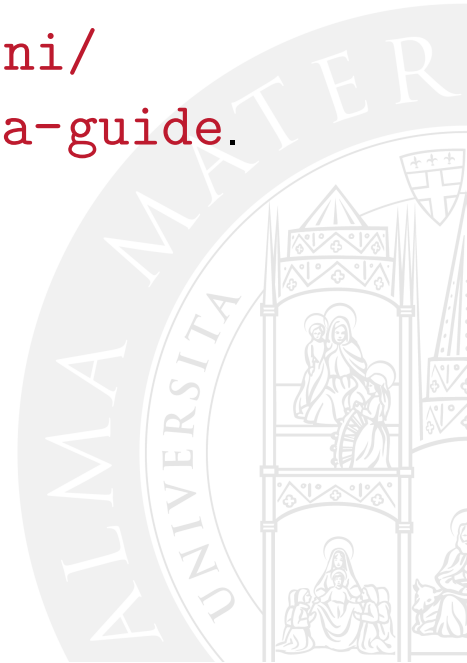
Stefano Mariani
s.mariani@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2014/2015

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.



Outline

Outline

Tuple Centres Spread over the Network (TuCSoN)

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

Nodes

- Each node within a TuCSoN system is *univocally identified* by the pair $< NetworkId, PortNo >$, 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

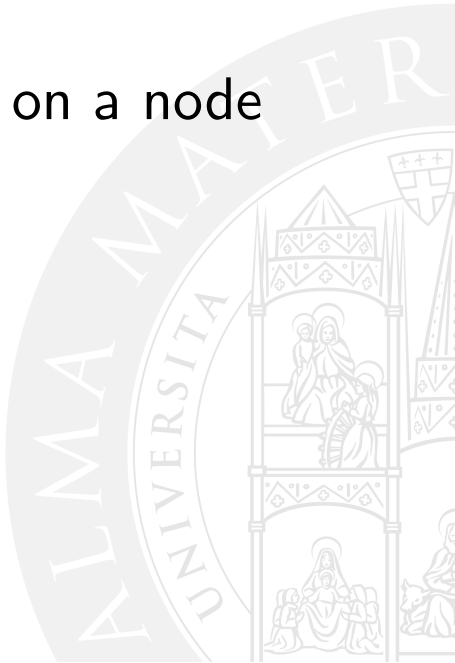
```
netid : portno
```

[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] [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

```
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
```

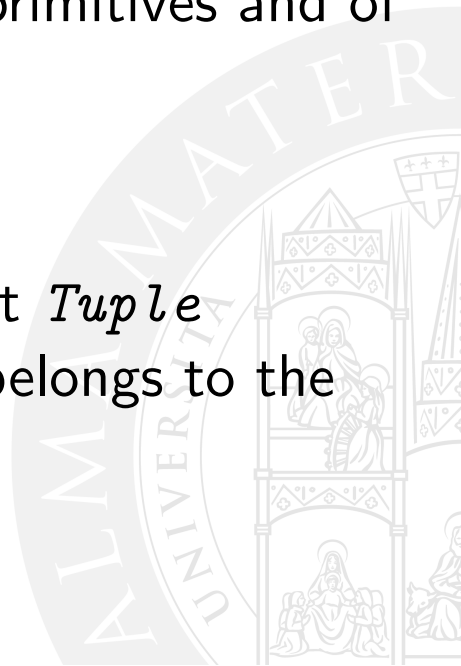
[3] `http://docs.oracle.com/javase/7/docs/api/java/util/UUID.html`

Outline

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



Tuple & Tuple Template Languages

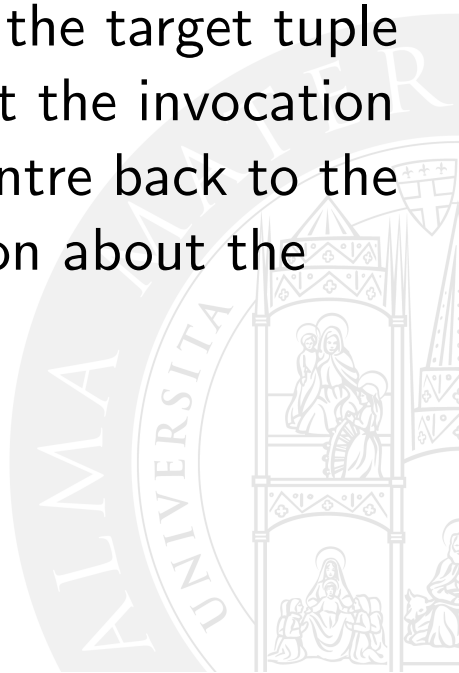
- Given that the TuCSoN coordination medium is the *logic-based* ReSpecT tuple centre, both the tuple and the tuple template languages are logic-based, too
- More precisely
  - any first-order logic Prolog atom is an admissible TuCSoN tuple...
  - ...and an admissible TuCSoN tuple template

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 `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

Coordination Primitives

The TuCSoN coordination language provides the following 9 basic[5] coordination primitives to build coordination operations

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

[5] We will see others in next lesson

Outline

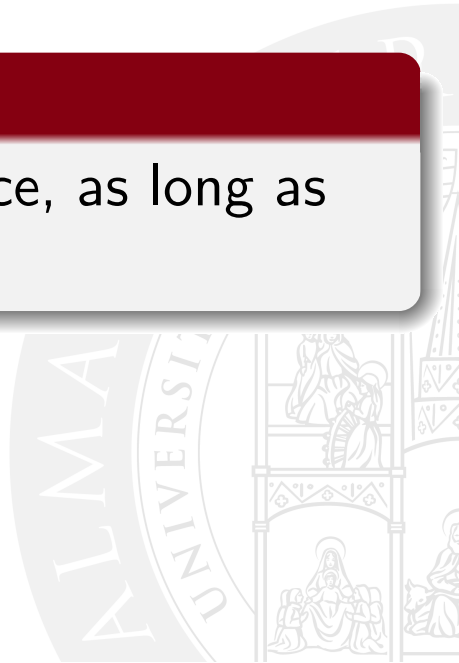
Outline

Node

- 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 `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] 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`

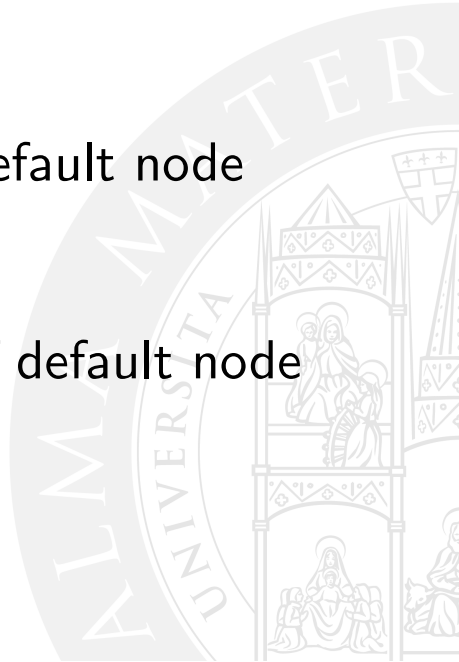
- As a result, agents can invoke operations of the form

  `@ netid : portno ? op`

  without specifying the tuple centre name `tname`

Defaults

- 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

Outline

Technology Requirements

- TuCSoN is a Java-based middleware (Java 7 is enough)
- TuCSoN is also Prolog-based[7]: 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

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

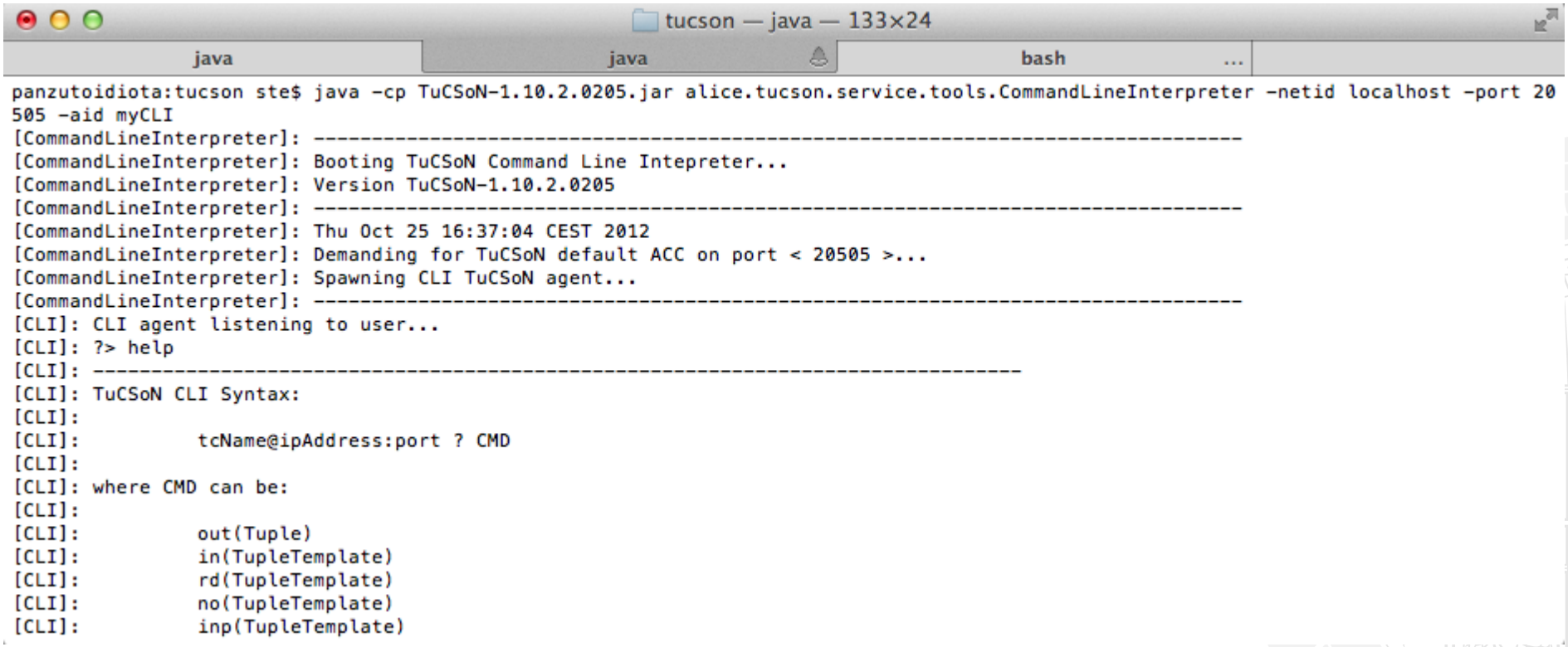
- The node service is in charge of
  - listening to incoming operation invocations
  - dispatching them to the target tuple centre
  - returning the operations completion to the source agent

Outline

Command Line Interpreter (CLI) I

- Shell interface for humans

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



Command Line Interpreter (CLI) II

CLI Syntax

⟨*TucsonOp*⟩	::=	⟨*TcName*⟩ `@` ⟨*IpAddress*⟩ `:` ⟨*PortNo*⟩ `?` ⟨*Op*⟩
⟨*TcName*⟩	::=	Prolog ground term
⟨*IpAddress*⟩	::=	`localhost` | IP address
⟨*PortNo*⟩	::=	port number
⟨*Op*⟩	::=	`out(T)` | `in(TT)` | `rd(TT)` | `no(TT)` |`inp(TT)` | `rdp(TT)` | `nop(TT)` | `get()` | `set([T1,...,Tn])` | `out_all(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`	::=	tuple (Prolog term)
`TT`	::=	tuple template (Prolog term)
`TL`	::=	list of tuples (Prolog list of terms)
`E,E1,...,En`	::=	ReSpecT event
`G,G1,...,Gn`	::=	ReSpecT guard predicate
`R,R1,...,Rn`	::=	ReSpecT reaction body
`ET`	::=	ReSpecT event template
`GT`	::=	ReSpecT guard template
`RT`	::=	ReSpecT reaction body template

Outline

TuCSoN Experiments I

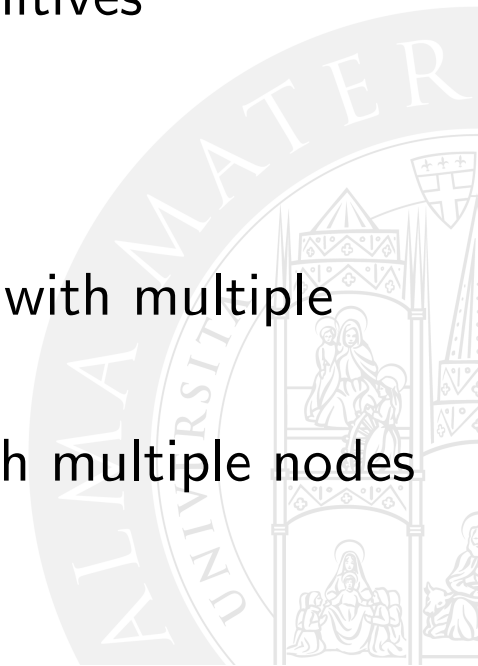
1. Launch a local TuCSoN Node

   ```
   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

Outline

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

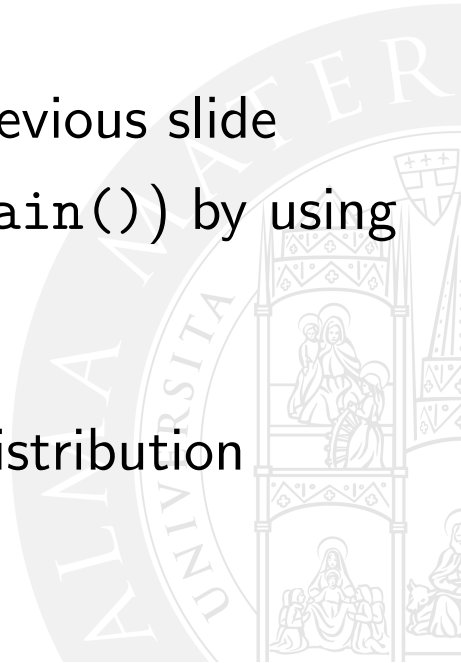
See example `HelloWorld` in package `alice.tucson.examples.helloWorld` within TuCSoN distribution

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

TuCSoN Experiments II

1. Launch a local TuCSoN Node

   ```
   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

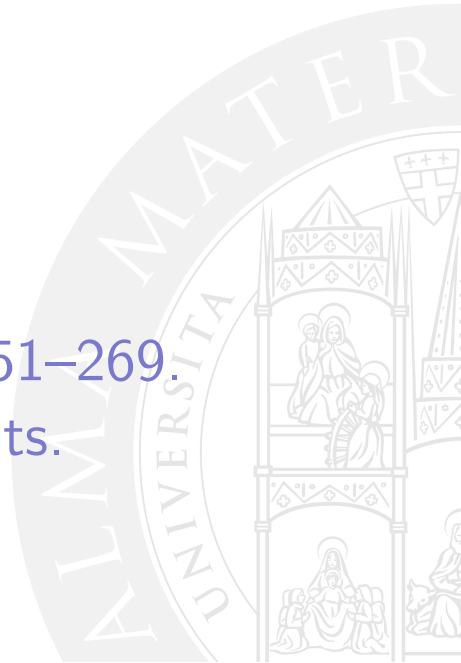
Outline

Bibliography


Lloyd, J. W. (1984).
Foundations of Logic Programming.
Springer, 1st edition.


Omicini, A. and Denti, E. (2001).
From tuple spaces to tuple centres.
Science of Computer Programming, 41(3):277–294.


Omicini, A. and Zambonelli, F. (1999).
Coordination for Internet application development.
Autonomous Agents and Multi-Agent Systems, 2(3):251–269.
Special Issue: Coordination Mechanisms for Web Agents.



TuCSoN: **Tu**ple **C**entres **S**pread **o**ver the **N**etwork 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