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

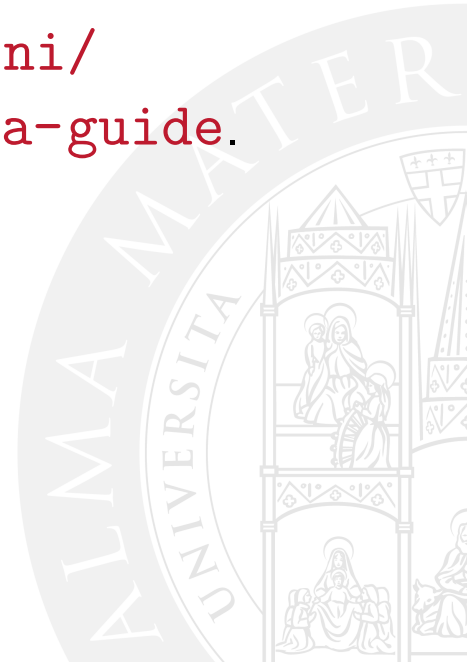
Stefano Mariani
s.mariani@unibo.it

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.

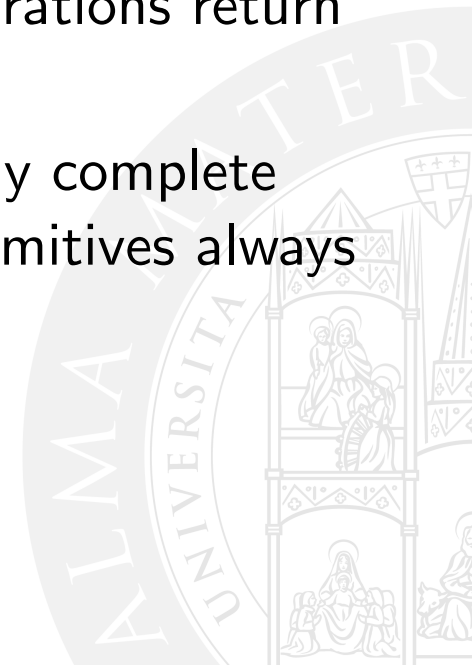


Outline

Outline

Bulk Primitives: The Idea

- Bulk coordination primitives provide significant efficiency gains for that class of coordination problems involving the management of multiple tuples using a *single coordination operation* [Rowstron, 1996]
- Briefly, instead of returning one single tuple, bulk operations return the *whole set of matching tuples*
- In case no matching tuples are found, they successfully complete anyway, returning an *empty list* of tuples (so, bulk primitives always succeed)



Bulk Primitives in TuCSoN

The TuCSoN coordination language provides the following 4 bulk coordination primitives:

`out_all(`*`Tuples`*`)` inserts in the target tuple space the given (Prolog) list of logic tuples

`rd_all(`*`Template`*`)` attempts to read from the target tuple space all the tuples matching the given template, returning them as a list (possibly empty)

`in_all(`*`Template`*`)` attempts to withdraw from the target tuple space all the tuples matching the given template, returning them as a list (possibly empty)

`no_all(`*`Template`*`)` tests the target tuple space for absence of any tuple matching the given template, returning the empty list in case of success and the whole set of matching tuples in case of failure

Bulk Primitives: CLI Experiments

Try bulk primitives vs. corresponding LINDA primitives:

- e.g., synchronise with M processes out of a pool of N (with $M < N$) in the most effective way;
- e.g., compute multiplicity of tuples or count how many tuples satisfy a given template;
- e.g., can any master-workers architecture benefit from these new primitives?

Outline

The spawn Primitive I

In order to enable TuCSoN agents to delegate complex computational activities related to coordination to the coordination medium itself, TuCSoN provides the spawn primitive—similaro to LINDA eval

Semantics

- spawn activates a *parallel computational activity* – actually, either a Java thread or a tuProlog engine – to be carried out asynchronously w.r.t. the caller—either an agent or the tuplecentre itself
- The execution of the spawn is local to the tuple space where it is invoked, and so are their results
  - correspondingly, the code (either Java or tuProlog) of the spawned computation must be local to the same node hosting the "spawning" tuple centre (no "code on demand")
  - also, the code can execute (a subset of) TuCSoN coordination primitives, but only on the same spawning tuple centre

The spawn Primitive II

General syntax

- spawn has basically two parameters
  - activity — a ground Prolog atom containing either the tuProlog theory along with the goal to be solved – e.g., `solve('path/to/Prolog/Theory.pl', yourGoal)` – or the Java class to be executed—e.g., `exec('list.of.packages.YourClass.class')`
  - tuple centre — a ground Prolog term identifying the target tuple centre that should execute the spawn
- From tuProlog, the two parameters are just the end of the story...

The spawn Primitive III

Java syntax

- . . . a third parameter is instead necessary when *spawning* from TuCSoN Java agent (homogeneously with other TuCSoN primitives)
- it could be either

  listener — a listener `TucsonOperationCompletionListener` in case of an asynchronous call of `spawn`

  timeout — an integer value in milliseconds determining the maximum waiting time for the agent in case of a synchronous call of `spawn`—notice its execution is still a separate, parallel computation

`spawn` primitive: CLI Experiments

Try to spawn a Java program as a parallel activity to be carried out by the coordination medium:

- e.g., coordinate 2 CLIs through the outcome of an expensive computation—or an expensive iteration over tuples in the space
- e.g., again, can any master-workers architecture benefit from this new primitives?

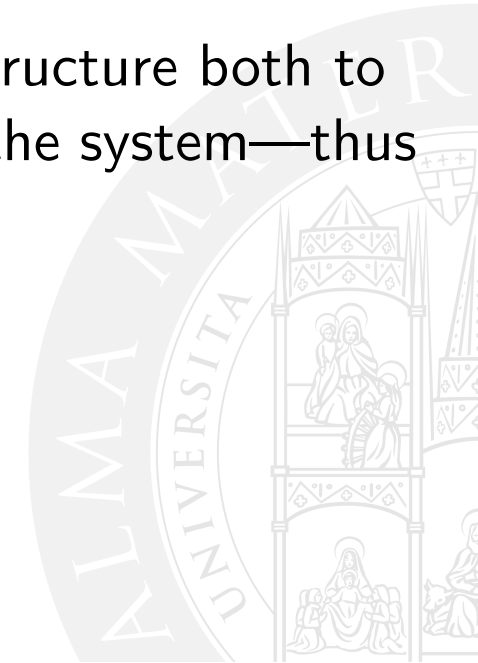
Outline

Outline

ACC

An Agent Coordination Context (ACC) [Omicini, 2002] is

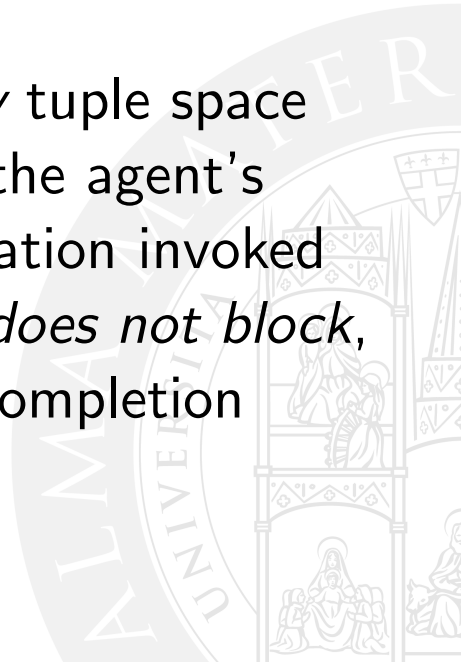
- a *runtime* and *stateful* interface released to an agent to execute operations on the tuple centres of a specific organisation
- a sort of *interface* provided to an agent by the infrastructure both to *enable and constraint* it admissible interactions with the system—thus other agents and the coordination medium itself



Ordinary ACCs

`OrdinarySynchACC` enables interaction with the *ordinary* tuple space and enacts a synchronous behaviour from the agent's perspective: whichever the coordination operation invoked (either suspensive or predicative), the proxy *blocks* waiting for its completion

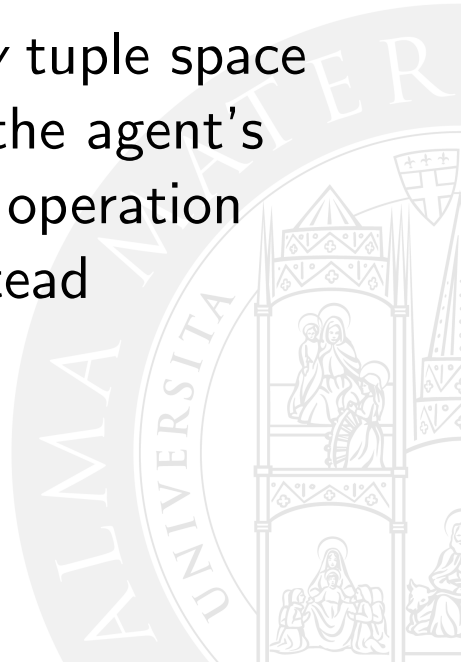
`OrdinaryAsynchACC` enables interaction with the *ordinary* tuple space and enacts an asynchronous behaviour from the agent's perspective: whichever the coordination operation invoked (either suspensive or predicative), the agent *does not block*, but is instead *asynchronously notified* upon completion



Bulk ACCs

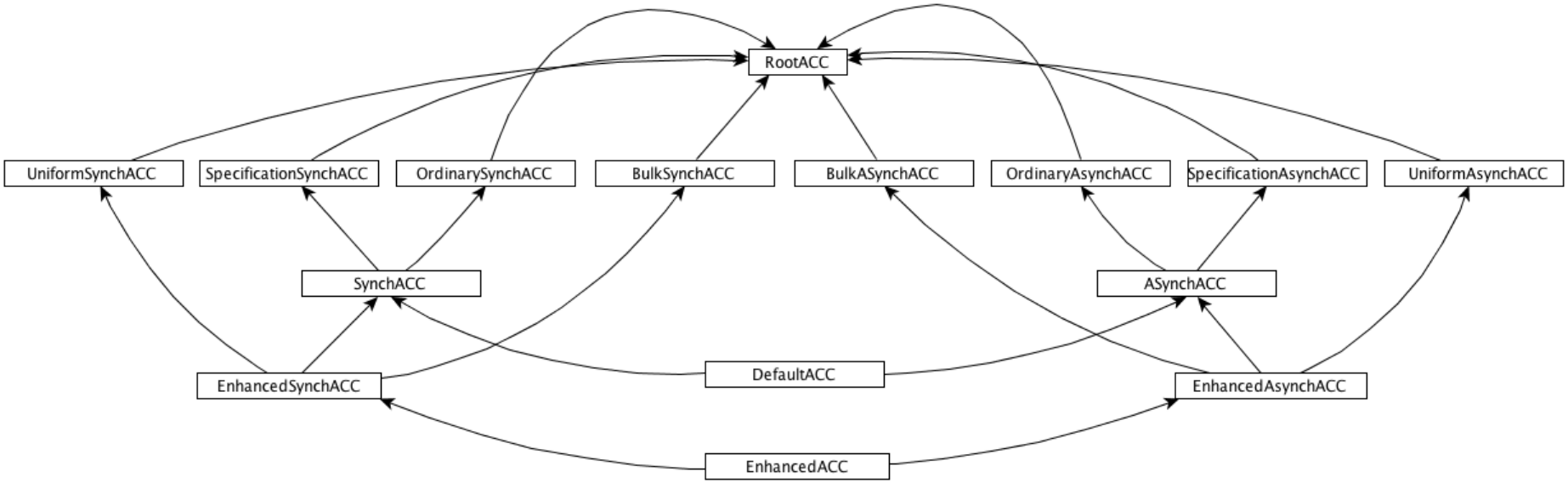
`BulkSynchACC` enables bulk interaction with the *ordinary* tuple space and enacts a synchronous behaviour from the agent's perspective: whichever the bulk coordination operation invoked, the agent *blocks* waiting for its completion

`BulkAsynchACC` enables bulk interaction with the *ordinary* tuple space and enacts an asynchronous behaviour from the agent's perspective: whichever the bulk coordination operation invoked, the agent *does not block*, but is instead *asynchronously notified* of its completion



Overall View over TuCSoN ACCs

Other ACCs exist: some enabling access to the ReSpecT specification space and others being a convenient combination of previous ones



Outline

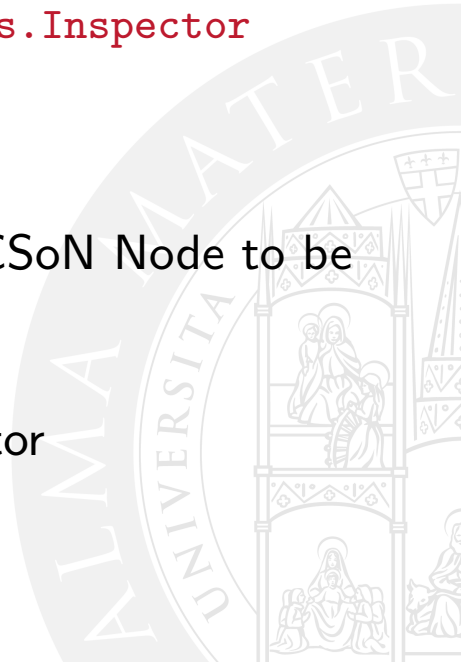
TuCSoN Inspector I

A GUI tool to monitor the TuCSoN coordination space & ReSpecT VM

- to launch the Inspector tool

```
java -cp tucson.jar:2p.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



TuCSoN Inspector II

What to inspect

In the *Sets* tab[a] 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

[a] The *StepMode* tab is for debugging of ReSpecT reactions.

Outline

Outline

Jade

- Jade is one of the oldest and nowadays most widely used agent development frameworks [Bellifemine et al., 2007]
- Jade can be downloaded freely from `http://http://jade.tilab.com`
- Integrating TuCSoN with Jade essentially means to make coordination via tuple centres generally available to agent programmers

TuCSoN4JADE

- TuCSoN4JADE integrate TuCSoN and JADE by implementing TuCSoN as a JADE *service* [Omicini et al., 2004]
- An example of how to use TuCSoN from JADE is reported in the TuCSoN main site at `http://apice.unibo.it/xwiki/bin/download/TuCSoN/Documents/tucson4jadequickguidepdf.pdf`

Synchronous vs. Asynchronous Invocation

- The `BridgeToTucson` class is the component mediating all the interactions between JADE and TuCSoN
- In particular, it offers two methods for invoking coordination operations, one for each *invocation semantics* JADE agents may choose [Mariani et al., 2014]:

  `synchronousInvocation()` — lets agents invoke TuCSoN coordination operations *synchronously w.r.t. the caller behaviour*. This means the caller behaviour *only* is (possibly) suspended – and automatically resumed – as soon as the requested operation completes, not the agent as a whole—as in [Omicini et al., 2004].

  `asynchronousInvocation()` — lets clients *asynchronously* invoke TuCSoN coordination operations. Regardless of whether the coordination operation suspends, the agent does not, thus the caller behaviour continues [Mariani et al., 2014].

Outline

Bibliography I

Bellifemine, F. L., Caire, G., and Greenwood, D. (2007).
Developing Multi-Agent Systems with JADE.
Wiley.

Mariani, S., Omicini, A., and Sangiorgi, L. (2014).
Models of autonomy and coordination: Integrating subjective & objective approaches in agent development frameworks.
In Braubach, L., Camacho, D., and Venticinque, S., editors, *8th International Symposium on Intelligent Distributed Computing (IDC 2014)*, Madrid, Spain.

Bibliography II

Omicini, A. (2002).
Towards a notion of agent coordination context.
In Marinescu, D. C. and Lee, C., editors, *Process Coordination and Ubiquitous Computing*, chapter 12, pages 187–200. CRC Press, Boca Raton, FL, USA.

Omicini, A., Ricci, A., Viroli, M., Cioffi, M., and Rimassa, G. (2004).
Multi-agent infrastructures for objective and subjective coordination.
Applied Artificial Intelligence: An International Journal, 18(9–10):815–831.
Special Issue: Best papers from EUMAS 2003: The 1st European Workshop on Multi-agent Systems.

Rowstron, A. I. T. (1996).
Bulk Primitives in Linda Run-Time Systems.
PhD thesis, The University of York.

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

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