

# TuCSoN: Tuple Centres Spread over the Network Advanced 2.0

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

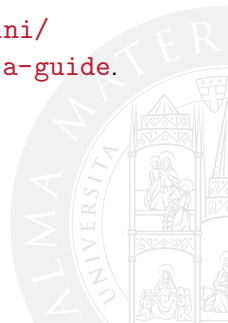


- 1 TuCSoN Asynchronous API
- 2 Objective + Subjective Coordination: TuCSoN4JADE
- 3 Access Control in TuCSoN: RBAC
- 4 Bibliography



# 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](http://www.slideshare.net/andreaomicini/the-tucson-coordination-model-technology-a-guide).



# Outline

- 1 TuCSoN Asynchronous API
- 2 Objective + Subjective Coordination: TuCSoN4JADE
- 3 Access Control in TuCSoN: RBAC
- 4 Bibliography



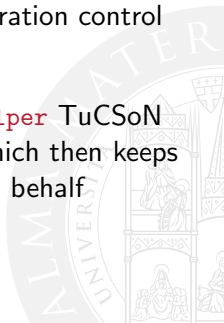
# Asynchronous Operation Invocation I

Coordination operations may be invoked in two modes

**synchronous** — blocking *the caller agent* whenever the invoked operation gets suspended

**asynchronous** — *preserving agents' own autonomy*, by *decoupling* the agent control flow from the coordination operation control flow

- ✓ Asynchronous mode is supported by the **AsynchOpsHelper** TuCSoN component in package `alice.tucson.asynchSupport`, which then keeps track of *pending* and *completed* operations on agents' behalf



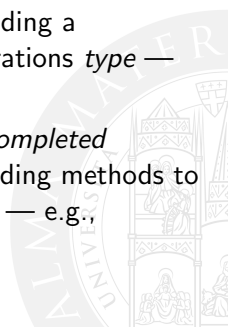
# Asynchronous Operation Invocation II

The API exposed by `AsyncOpsHelper` consists of

`enqueue(AbstractTucsonAction, TucsonOperationCompletionListener): boolean` — adds an operation to the queue of *pending operations*, given the listener component to notify upon its completion

`getPendingOps(): SearchableOpsQueue` — gets the queue of pending operations, that is, a *thread-safe* queue providing a `getMatchingOps(...)` method to *filter* on operations *type* — e.g., *in*, *rd*, etc.

`getCompletedOps(): CompletedOpsQueue` — gets the queue of *completed operations*, that is, a *thread-safe* queue providing methods to *filter* on operations *features* (*type*, *outcome*) — e.g., *successful operations*, *failed operations*



# Asynchronous Operation Invocation III

`shutdownGracefully(): void` — requests *soft shutdown* of the helper, that is, shutdown *waits* for pending operations to complete

`shutdownNow(): void` — requests *hard shutdown* of the helper, that is, shutdown happens as soon as the currently executing operation completes — other pending operations are *discarded*

## Further Reference

More details in “Asynchronous Operation Invocation in TuCSoN” how-to at <http://apice.unibo.it/xwiki/bin/view/TuCSoN/Documents>

## Let's try!

Check out example `PrimeCalculationLauncher` in package `ds.lab.tucson.asyncAPI`

# Outline

- 1 TuCSoN Asynchronous API
- 2 Objective + Subjective Coordination: TuCSoN4JADE**
- 3 Access Control in TuCSoN: RBAC
- 4 Bibliography



# Context

- In **objective coordination**, coordination-related concerns are extracted from agents to be embodied within dedicated abstractions offering *coordination as a service* [VO06]
- In **subjective coordination** instead, coordination issues are directly tackled by individual agents themselves

## Objective & Subjective Coordination [OO03]

Objective and subjective coordination thus constitute two *complementary* approaches, *both* essential in MAS design and development [ROD03], hence requiring a suitable **integration**



# Motivation

- Successful integration depends on the *technology level*, that is, on the mechanisms provided by the agent frameworks to be integrated
- In particular, it depends on the **model of autonomy** promoted by the specific agent platform, and by its relationship with the **model of coordination** adopted by the specific (objective) coordination framework

## Hindering Autonomy

Any integration effort *not* taking into account such two aspects is likely to **hinder agent autonomy** by (unintentionally) creating *artificial dependencies* between the subjective and the objective stances on coordination



# The Issue of Autonomy I

## Model of Autonomy

A model defining

- how agents behave as *individual* entities
- how they relate to each other as *social* entities
- how the two things *coexist*

## Model of Coordination

A model defining the semantics of the admissible *interactions* between agents in a MAS, in particular, w.r.t. their effects on the agent autonomy (e.g., *control flow*)



# The Issue of Autonomy II

## JADE Model of Autonomy

- Behaviours for individual tasks
- Asynchronous messages for subjective coordination
- The “block()-then-resume” pattern to reconcile individual and social attitudes



## The Issue of Autonomy III

ACCs are fundamental to *guarantee and preserve* agent autonomy by enabling separation of the **suspensive semantics** of a coordination operation from its **invocation semantics**

### Operation Execution

- invocation** — the *request* to carry out a given coordination operation is sent to TuCSoN
- completion** — the *response* to the coordination operation invoked is sent back to the requesting agent

### TuCSoN Model of Coordination

By decoupling invocation semantics from the operation semantics, synchronous calls are always consequence of the *agent own deliberation* process

# TuCSoN4JADE I



Home Documentation Support Download Developers News Related Projects

## Third Party Contributions

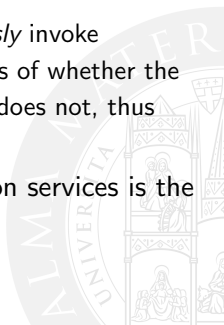
This page contains JADE-related software developed by various users from the JADE community.

The shown information is that reported by the developers themselves. The JADE team is not responsible for the correctness of this information neither for the legal correctness of the license.

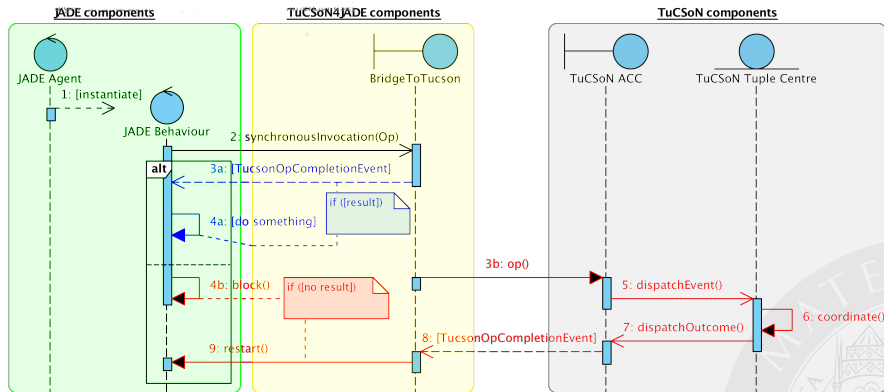
| File        | Since      | Author          | License | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|------------|-----------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TuCSoN4JADE | JADE 4.3.2 | Stefano Mariani | LGPL    | <p>TuCSoN4JADE (T4J for short) is a Java library enabling JADE agents to exploit TuCSoN coordination services wrapped as an ad-hoc API into a JADE kernel service.</p> <p>TuCSoN is a Java-based middleware providing software agents with coordination as a service via programmable logic tuple spaces, called tuple centres. By combining TuCSoN and JADE, T4J aims at providing MAS engineers with a full-featured MAS middleware, enabling them to exploit both dimensions of agent-oriented software engineering (individual, through JADE agents; social, via TuCSoN tuple centres) in a complete and well-balanced way.</p> |

# TuCSoN4JADE II

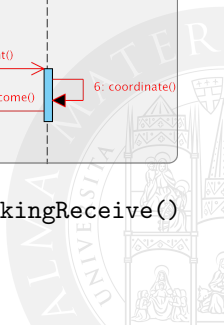
- TuCSoN4JADE preserves autonomy of agents by providing them with two invocation semantics regarding coordination services:
  - `synchronousInvocation()` — lets agents invoke TuCSoN coordination operations *synchronously w.r.t. the caller behaviour*. This means the caller behaviour *only* is (possibly) suspended and automatically resumed.
  - `asynchronousInvocation()` — lets clients *asynchronously* invoke TuCSoN coordination operations. Regardless of whether the coordination operation suspends, the agent does not, thus the caller behaviour continues.
- The method for synchronous invocation of coordination services is the one we are interested in



# TuCSon4JADE III



**Figure:** The “alt”-labelled frame is the equivalent of JADE `blockingReceive()` programming pattern in TuCSon4JADE.



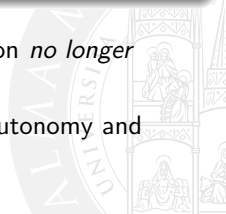
# TuCSoN4JADE IV

- We know when JADE behaviour is re-scheduled, its `action()` method re-starts *from the beginning*, thus, method `synchronousInvocation()` is re-invoked

## Autonomy Unhindered

The whole TuCSoN4JADE machinery works because such method internally (thus *transparently*) checks if operation completion is already available: only if it is not, the behaviour (**only**) gets suspended, thus the whole path 3.b-9 executed.

- This way, the agents choice to rely on objective coordination *no longer affects* their ongoing subjective coordination activities
- This has been possible by accounting for JADE model of autonomy and TuCSoN model of coordination while planning integration



# Scenario I

- $n$  seller agents advertise their catalogue of books
- $m$  buyer agents browse such catalogues looking for books
- The whole interactions chain has the form of the well-known *ContractNet protocol*:
  - ① buyers start a call-for-proposals
  - ② sellers reply with actual proposals
  - ③ buyers choose which one to accept
  - ④ the purchase is carried out

## Concurrency Property

Sellers should stay reactive to call-for-proposals even in the middle of a purchase transaction—otherwise they could lose potential revenues. We call **concurrency property** such a requirement.

## Scenario II

- We re-think the ContractNet protocol by integrating objective and subjective coordination: **tuple-based call-for-proposals** (thus, objective coordination) with **message-based purchase** (hence, subjective coordination)
- The call-for-proposals should reach all the sellers, thus it is more efficient to put a single “call-for-proposals tuple” in a shared “contract-net space”, rather than messaging each seller individually
- The purchase is typically a 1-to-1 interaction, hence messaging can efficiently do the job
- As expected, the concurrency property is preserved



## Scenario III

### Let's Try!

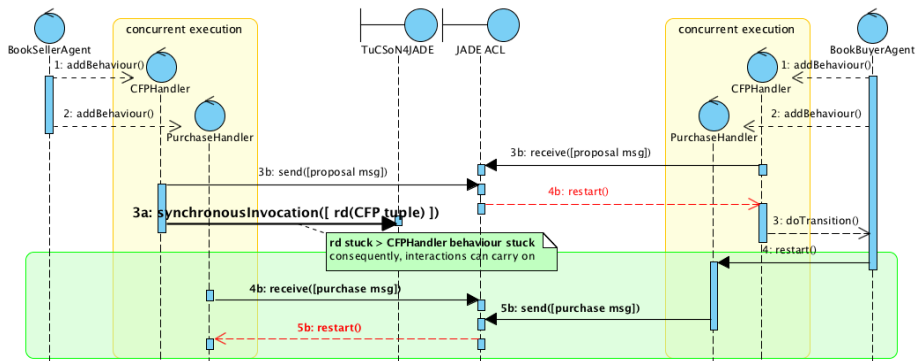
Check out “Book trading” example in package

`ds.lab.tucson.forjade.bookTrading`

- 1 Launch JADE platform with additional option `-services it.unibo.tucson.jade.service.TucsonService`
- 2 From JADE RMA GUI launch *one* BookSellerAgent
- 3 From JADE RMA GUI launch any number of BookBuyerAgents
- 4 Shutdown all buyers
- 5 Shutdown the seller



# TuCSoN4JADE Approach



**Figure:** The **rd** suspensive semantics is confined to the caller behaviour, thus **only** the caller behaviour is suspended, whereas other activities can carry on concurrently—e.g., the purchase transaction already in place (4b-5b).

# Further References

## URLs

- <http://tucson4jade.apice.unibo.it/>
- <http://jade.tilab.com/download/third-party-contributions/>

## Codebase

- <https://bitbucket.org/smariani/tucson4jade>

## Paper [MOS14]

- [http://link.springer.com/10.1007/978-3-319-10422-5\\_9](http://link.springer.com/10.1007/978-3-319-10422-5_9)



# Outline

- 1 TuCSoN Asynchronous API
- 2 Objective + Subjective Coordination: TuCSoN4JADE
- 3 Access Control in TuCSoN: RBAC**
- 4 Bibliography



# RBAC API I

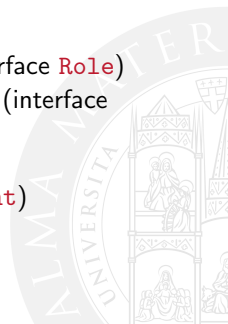
- Interface **RBACStructure**

- implementation class `TucsonRBACStructure`
- package `alice.tucson.rbac`

models a RBAC organisation within TuCSoN

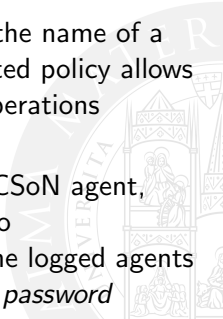
- It includes

- a set of **roles**, as instances of class `TucsonRole` (interface **Role**)
- a set of **policies**, as instances of class `TucsonPolicy` (interface **Policy**)
- a set of **authorised agents**, as instances of class `TucsonAuthorisedAgent` (interface **AuthorisedAgent**)



# RBAC API II

- Class `TucsonRole` includes, besides its name and description:
  - the *policy* it adheres to
  - the **agent class** associated to the role, allowing activation of the role only for those agents belonging to such class
- Class `TucsonPolicy` includes, besides its name:
  - a set of **permissions**, as instances of class `TucsonPermission` (interface **Permission**)
- Class `TucsonPermission`, currently, simply represents the name of a TuCSoN primitive, to model the fact that the associated policy allows agents with the associated role to request TuCSoN operations involving that primitive
- Class `TucsonAuthorisedAgent` models a *recognised* TuCSoN agent, that is, an agent who performed a successful *login* into RBAC-TuCSoN; as such, it includes the *agent class* the logged agents belongs to, its (encrypted) *username* and (encrypted) *password*



## RBAC API III

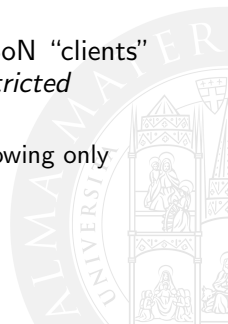
### Further reference

Other RBAC-related properties belonging to the TuCSoN node – hence to `TucsonNodeService` class – can be configured — see “RBAC in TuCSoN” how-to at <http://apice.unibo.it/xwiki/bin/view/TuCSoN/Documents>



# RBAC API IV

- To participate a TuCSoN-RBAC organisation, agents need to
  - ① acquire a **meta-ACC**
  - ② *activate* a role to acquire an **ACC**
- Step 1 involves class **TucsonMetaACC**, within package `alice.tucson.api`:
- Step 2 involves the **NegotiationACC**, which lets TuCSoN “clients” acquire an ACC, by *playing* RBAC roles, enabling *restricted* interaction with TuCSoN coordination services
  - ✓ the released ACC is equipped with a *built-in filter* allowing only **admissible operations** according to the agent's role



# RBAC API V

`playRole(String, Long): EnhancedACC` — attempts to play the given role

`playRoleWithPermissions(List<String>, Long): EnhancedACC` — attempts to play a role given a set of *desired permissions*. The principle according to which a role is selected is the **least privilege**: among the roles enabling all desired permissions, the one giving the least permissions is selected—if no suitable role is found, no ACC is released

Let's try!

Check out example `RBACLauncher` in package `ds.lab.tucson.rbac`



# Outline

- 1 TuCSoN Asynchronous API
- 2 Objective + Subjective Coordination: TuCSoN4JADE
- 3 Access Control in TuCSoN: RBAC
- 4 Bibliography**



# Bibliography I



Stefano Mariani, Andrea Omicini, and Luca Sangiorgi.

Models of autonomy and coordination: Integrating subjective & objective approaches in agent development frameworks.

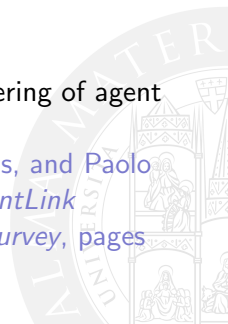
In Lars Braubach, David Camacho, and Salvatore Venticinque, editors, *8th International Symposium on Intelligent Distributed Computing (IDC 2014)*, Madrid, Spain, 3–5 September 2014.



Andrea Omicini and Sascha Ossowski.

Objective versus subjective coordination in the engineering of agent systems.

In Matthias Klusch, Sonia Bergamaschi, Peter Edwards, and Paolo Petta, editors, *Intelligent Information Agents: An AgentLink Perspective*, volume 2586 of *LNAI: State-of-the-Art Survey*, pages 179–202. Springer, 2003.



## Bibliography II



Alessandro Ricci, Andrea Omicini, and Enrico Denti.

Activity Theory as a framework for MAS coordination.

In Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *LNCS*, pages 96–110. Springer, April 2003.



Mirko Viroli and Andrea Omicini.

Coordination as a service.

*Fundamenta Informaticae*, 73(4):507–534, 2006.

Special Issue: Best papers of FOCLASA 2002.



# TuCSoN: Tuple Centres Spread over the Network Advanced 2.0

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

