

RBAC in TuCSoN

In this brief "how-to", you will learn TuCSoN main API available to developers of MAS willing to exploit the *RBAC (Role-Based Access Control) model* in their TuCSoN-coordinated system.

Assumptions are you are familiar with TuCSoN core API and with basic notions of the RBAC model.

Suggested readings complementing the content of this "how-to" are:

- Omicini, A., Ricci, A., Viroli, M.: "*RBAC for Organisation and Security in an Agent Coordination Infrastructure*". In Electronic Notes in Theoretical Computer Science 128(5), pages 65-85, 3 May 2005. Available [here](#).

1. [Role-Based Access Control in TuCSoN](#)

1.1 [API Overview](#)

1.2 ["Hands-on" step-by-step tour](#)

2. [Contact Information](#)

1. Role-Based Access Control in TuCSoN

1.1 API Overview

>>>**RBAC properties.** Interface `RBACStructure`, along with implementation class `TucsonRBACStructure`, both within package `alice.tucson.rbac`, models a **RBAC organisation** within TuCSoN. It includes, besides the name of the organisation and a few configuration parameters:

- a set of **roles**, as instances of class `TucsonRole` within the same package, whose interface is `Role`
- a set of **policies**, as instances of class `TucsonPolicy` within the same package, whose interface is `Policy`
- a set of **authorised agents**, as instances of class `TucsonAuthorisedAgent` within the same package, whose interface is `AuthorisedAgent`

Class `TucsonRole` models a RBAC role within TuCSoN. It includes, besides its name and description:

- the policy it adheres to, defining the permissions attributed to the role
- the **agent class** associated to the role, allowing activation of the role only for those

agents belonging to such class

Class `TucsonPolicy` models a RBAC policy within TuCSoN. It includes, besides its name, a set of **permissions**, as instances of class `TucsonPermission` within the same package (interface `Permission`), modeling the TuCSoN operations allowed by this policy.

Class `TucsonPermission` models a RBAC permission within TuCSoN. Atm, a TuCSoN permission simply is 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.

Besides such RBAC properties, other properties belonging to the TuCSoN Node -- hence `TucsonNodeService` class -- can be configured to increase RBAC customisation opportunities:

```
void setListAllRolesAllowed(boolean)
```

to allow agents to query the RBAC structure installed in TuCSoN for a list of those roles available to them.

```
void setAdminUsername(String)
void setAdminPassword(String)
```

to configure credentials for **administrators** of RBAC-TuCSoN, that is, those agents who are allowed to modify the RBAC structure installed in TuCSoN.

```
void setInspectorsAuthorised(boolean)
```

to allow inspection of tuple centre `'$ORG'` to TuCSoN **Inspector tool**. Such tuple centre is the one storing RBAC-related tuples and reactions, thus forbidding inspection enhances security.

```
void setLoginRequired(boolean)
```

to constrain access to the TuCSoN-coordinated system only to logged (recognised) agents. Otherwise, also non-logged agents may participate the system, although with limited access to TuCSoN coordination services--according to the roles associated to the *basic agent class*.

```
void setBasicAgentClass(String)
```

to configure the basic agent class automatically associated to non-logged agents.

>>>RBAC interaction. To participate a RBAC organisation installed in TuCSoN, 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`. Depending on whether the requesting agent wants a meta-ACC for **administration** or **negotiation** purpose, two methods are provided:

```
AdminACC getAdminContext(TucsonAgentId, String, int, String, String)

NegotiationACC getNegotiationContext(TucsonAgentId, String, int)
```

The first one returns to the caller agent an `AdminACC`, provided it gives a valid TuCSoN agent ID, and valid login credentials (last two parameters, encrypted)--besides issuing the correct IP address / TCP port number combination where the TuCSoN Node is active (second and third parameters).

The second one returns to the caller a `NegotiationACC`, provided a valid TuCSoN agent ID, and the correct IP address / TCP port number combination are given.

An `AdminACC` lets system administrators manage the RBAC configuration of TuCSoN at run-time by providing the following methods--shielding access to the special tuple centre '`$ORG`' devoted to RBAC configuration management:

```
void install(RBACStructure, Long, String, int)
```

Installs a pre-configured RBAC structure in the target TuCSoN node--the one identified by the IP/TCP address given by last two parameters (the `Long` argument is a timeout). Essentially, installation involves adding roles, policies, authorised agents, and all the RBAC-related properties of the TuCSoN node--the basic agent class, whether login is required or not, whether inspection is allowed or not.

```
void removeRBAC(Long, String, int)
```

Removes the RBAC installation from the given TuCSoN node---parameters as previous method.

```
void add(Role)
```

Adds a new role to the RBAC configuration installed in default TuCSoN node (that is, reachable at localhost:20504). Addition of a new policy including such role should follow for the new role to be meaningful.

```
void removeRole(String)
```

Removes the given role from the RBAC configuration installed in default TuCSoN node. Removal of the role from any possible associated policy is automatic.

```
void add(AuthorisedAgent)
```

Adds an authorised agent to the RBAC configuration installed in default TuCSoN node.

```
void setRolePolicy(String, String)
```

Adds an association between a policy (second arg) and a role (first arg), provided they both exist, to the RBAC configuration installed in default TuCSoN node.

```
void add(Permission, String)
```

Adds a permission to the given policy, provided they both exist, to the RBAC configuration installed in default TuCSoN node.

```
void setRoleAgentClass(String, String)
```

Sets the agent class (second arg) associated to a given already existing role (first arg) in the RBAC configuration installed in default TuCSoN node.

```
void setBasicAgentClass(String)
```

Sets the basic agent class in the RBAC structure installed in default TuCSoN node. Existing agents descriptions and opened sessions are automatically updated accordingly.

Step 2 involves the **NegotiationACC**, which lets TuCSoN clients **play RBAC roles**, so as to acquire an ACC enabling *restricted interaction* with TuCSoN coordination services according to the RBAC model--the released ACC is configured with a built-in *filter* allowing only

admissible operations according to the agents' role. The request to play a role may be sent through two different methods:

```
EnhancedACC playRole(String, Long)
```

```
EnhancedACC playRoleWithPermissions(List<String>, Long)
```

The first method attempts to play the given role (first parameter), according to the RBAC configuration installed in the TuCSoN node who released this negotiation ACC, waiting the given timeout (second parameter, in milliseconds) at most for operation completion.

The second one attempts to play a role given a set of desired permissions (first parameter), according to the RBAC configuration installed in the TuCSoN node who released this negotiation ACC, waiting the given timeout (second parameter, in milliseconds) at most for operation completion. The principle according to which a role is selected, is the **least privilege**: among the roles enabling *all* the desired permissions, the one giving the *least* permissions is selected--if no role is found allowing all the desired permissions, no ACC is released.

Other methods provided by `NegotiationACC` interface include:

```
EnhancedACC playDefaultRole()
```

to play the **default role**, which allows *any* TuCSoN operation *if and only if* no RBAC installation is available; in case RBAC is installed, instead, no TuCSoN operation is allowed--actually, no ACC is released at all.

```
boolean login(String, String)
```

to attempt a login given the username and password (first and second parameter, respectively), so as to receive the associated agent class according to RBAC configuration--necessary to activate the roles associated to that class.

```
List<Role> listPlayableRoles()
```

to have the list of the roles currently playable by the requesting agent, according to RBAC configuration and to the class of the requesting agent. This operation may be allowed or not depending on TuCSoN node configuration of RBAC-related properties (see above, under "RBAC properties").

1.2 "Hands-on" step-by-step tour

TBD.

In the meanwhile, you can read through the comments in TuCSoN "rbac" example source files, [here](#).

Contact Information

Author of this "how-to":

- *Stefano Mariani*, DISI - Università di Bologna (s.mariani@unibo.it)

Authors of TuCSoN in TuCSoN "People" section of its main site, here >
<http://apice.unibo.it/xwiki/bin/view/TuCSoN/People>.

[1]