The TENCompetence approach to authorization Version 0.1

Juri L. De Coi

December 21, 2008

Abstract

This report is the first version of a document which is meant to track the progress in authorization-related issues within TENCompetence. It contains (i) an introduction to the PROTUNE framework (ii) a TENCompetence authorization scenario and (iii) the current state of the implementation.

1 The Protune framework

This section introduces some features of the Protune framework whose application to TENCompetence may be desirable. Such features are

- the expressiveness of the Protune policy language
- the way the Protune framework evaluates policies
- the natural language-based user interface

1.1 The Protune policy language

The Protune framework allows users to define and enforce *policies*. An example authorization policy (in natural language) follows.

John is allowed to read "myFile" each day from 9:00 to 17:00.

While not that meaningful, the example above helps in identifying the different elements a generic policy consists of

actor the entity (not) allowed to perform some action on some resource (e.g., "John")

action which kind of access some actor has (not) on some resource (e.g., "read")

resource the entity some action is (not) allowed to be performed on (e.g., "myFile")

environmental constraints further conditions which do not depend on actor, action or resource (e.g., "each day from 9:00 to 17:00")

Protune policies do not need to explicitly mention actor, action, resource and environmental constraints since the Protune language allows to identify them by means of their properties, like in the following example.

> Every authenticated user is allowed to access the files contained in "myDirectory" according to her access rights and only during working time.

Again this policy involves actors, actions, resources and environmental constraints, but this time they are not referenced by name but by their properties

actor every authenticated user

action the actions the user has rights for

resource the files contained in "myDirectory"

environmental constraints during working time

Finally notice that, as the item "action" shows, the expressiveness of the Protune language allows to define complex conditions involving not only either of actors, actions, resources and environmental constraints but also all of them (since access rights differ from person to person and from resource to resource).

1.2 Policy evaluation

In order to enforce the policy mentioned above the Protune engine must (of course) be able to get information about

- whether a user has been authenticated
- which rights a user has on a resource
- which are the files contained in "myDirectory"
- what "working time" means

Retrieving this information may require to perform some actions (e.g., in order to find out which files are contained in "myDirectory" the Protune engine must access the file system). The Protune framework provides a means to smoothly plug actions into it.

1.3 The natural language-based user interface

The ultimate goal of Protune is to use natural language for every interaction with the user, i.e.

1. for policy definition
2. for answering policy-related questions (e.g., for explaining the decisions taken by the Protune engine)

The first item is a current research issue. So far we support relatively complex policies like the following one.

> Every user who sends a credential
>
> - that is valid and
> - whose type is "creditCard" and
> - whose owner is authenticated and
> - on which a price is charged
>
> pays the price with "creditCard".

Nevertheless further work is needed to support users in the task of defining policies (and especially to constrain them to only define – syntactically – correct policies).

The second item has been already addressed: the Protune framework supports HOW-TO, WHAT-IF, WHY and WHY-NOT natural-language explanations which allow a user to get information about (respectively)

- which actions she has to perform in order to access some resource
- how the Protune engine would behave assuming a given user behavior
- why the Protune engine allowed her to access some resource
- why the Protune engine did not allow her to access some resource

An example WHY-NOT explanation follows.

> I cannot prove that it is allowed to retrieve "myFile" because:
>
> 1. Rule [0] cannot be applied:
>   - it is not the case that "myFile" is a public document [details]
> 2. Rule [1] cannot be applied:
>   - "myFile" is a protected document [details]
>   - entities certified by Visa can access "myFile" [details]
>
>   but

    - it is not the case that a credential with issuer Visa has been received. [details]

The “details” tags are links to more fine-grained explanations, so that the explanation tree can be navigated by clicking on them. E.g., by clicking on the first “details” tag a new page opens showing why “myFile” is not a public document.

2 A TENCompetence authorization scenario

This section describes how the Protune framework can address the TENCompetence authorization scenario discussed in `http://www.partners.tencompetence.org/mod/forum/discuss.php?d=2160`.

2.1 Scenario

This section describes the entities the scenario introduces, namely resources, learners and services.

With the term “resource” we mean UOLs, learning activities, competence profiles, CDPs, communities and in general whatever entity can be accessed. Resources can refer to each other and can be either intermediate or final. Intermediate resources can be read or written (since the TENCompetence system is also a – collaborative – editing environment), whereas public resources can only be read. Finally, intermediate resources can become final ones, but final resources cannot become intermediate ones.

Learners are described by means of learner profiles (one per community) containing at least

- learnerId
- name
- email
- learning goal(s), i.e., selected competence profile(s) and selected CDP(s)
- for each selected CDP and for each competence in such CDP
  - already acquired competences
  - level of acquired competences
  - competences the learner is currently working on
- for each selected CDP and for each learning activity in the CDP
  - starting date
  - completion date
  - proficiency level
  - (if applicable) quiz score

- with whom the learner has contacts and which type of contacts

Services may require to access learner data and may require learners to share some profile data with others. Examples of services are visualization services, PCM services and the social help service.

2.2 Requirements

This section describes the security requirements which must be supported according to the different kinds of resources, namely intermediate resources, final resources, log data and learner profiles.

Intermediate resources the creator of an intermediate resource must be able to decide who is allowed to collaboratively edit it

Final resources the user adding a final resource must be able to decide who is allowed to access it

Log data (e.g., who added or viewed some resource, completed some activity, performed some action) can only be made available anonymously, except in case the learner explicitly approved

Learner profiles learners may allow services to access (part of) them

2.3 Design

This section lists the places in the scenario where policies may play a role.

- As soon as an intermediate resource is created, a policy should be defined specifying who may edit it
- As soon as a final resource is added
  - either a policy is explicitly defined specifying who may access it
  - or the final resource is considered to be public
- At which granularity level (e.g., whole system, CDP, competence) should users define whether and which log data they want to make available?
  - either users explicitly define policies specifying which log data they want to make available
  - or it is assumed that log data can only be made available anonymously
- As soon as a learner wants to use a new service, a policy should be defined specifying which parts of the learner profile as well as which log data the service may access
  - Services must provide terms of use specifying which data they will use and for which purpose

Notice that all policies introduced in this section can be straightforwardly expressed in Protune.

3 State of the implementation

The main authorization-related tasks of L3S within TENCompetence are

1. to identify and implement the actions which the TENCompetence scenarios require to support (cf. Section 1.2)
2. to provide a user-friendly natural language-based interface for every interaction with the user (cf. Section 1.3)
3. to make such services available within the TENCompetence infrastructure

The first point requires support from other TENCompetence partners in order to

- identify the scenarios which need to be supported and extract the requirements from them
- eventually implement the corresponding actions the Protune engine needs to be aware of

The current state of the second point has been reported in Section 1.3.

The third point has been addressed by providing a Web Service interface to the Protune framework which TENCompetence applications may use for authorization purposes. The remaining of this section describes the Web Service interface.

3.1 Web Service interface

This section provides a general overview of the Web Service interface. More detailed information can be found in the javadoc documentation of the service.

Tab. 3.1 provides an overview of the functionalities supplied by the TENCompetence authorization service. Such functionalities will be thoroughly described and exemplified in the following. Although the examples will show how the authorization service can be used over a GET protocol, the POST protocol is supported as well (and even exploited in the current implementation).

3.1.1 addResource

You can invoke method `addResource` as follows

> serviceURL [1]/addResource?resourceId=**resourceId**&repositoryId=**repositoryId**&ownerId=**ownerId**&securityLevel=**securityLevel**

[1]Here and in the following with "serviceURL" URL `http://10c.l3s.uni-hannover.de:9080/axis2/services/AuthService` is meant.

addResource	Add owner and security level (cf. below) of a (newly-created) resource to the policy knowledge base
deleteResource	Removes a resource from the policy knowledge base
update	Updates the security level of a resource
exists	Checks whether a resource is available in the policy knowledge base
getSecurityLevel	Returns the security level of a resource
getResources	Returns the resource ids of the resources having a given security level
selectAllowedResources	Selects the subset of resources a user can access out of a set of given resources
commit	Commits the last changes eventually performed to the policy knowledge base

Table 1: Overview of the functionalities supplied by the TENCompetence authorization service

where

resourceId is the id of a resource a user just created

repositoryId is the id of the repository where the resource is stored

ownerId is the id of the user who just created the resource

securityLevel represents the security level of the resource. Can be either 0 (private) or 1 (public) or 2 (only for logged-in users) or 3 (only for members of the same group)

The result will be . . .

```
<ns:addResourceResponse>
<ns:return/>
</ns:addResourceResponse>
```

. . . if everything went fine or . . .

<ns:addResourceResponse>
<ns:return>
errorDescription
</ns:return>
</ns:addResourceResponse>

. . . if something went wrong, where "errorDescription" is the description of the error occurred (typically the trace of the thrown exception).

3.1.2 deleteResource

You can invoke method `deleteResource` as follows

> serviceURL/deleteResource?resourceId=**resourceId**&repositoryId=
> **repositoryId**

where

resourceId is the id of the resource to be removed

repositoryId is the id of the repository where the resource is stored

The result will be . . .

> <ns:deleteResourceResponse>
> <ns:return/>
> </ns:deleteResourceResponse>

. . . if everything went fine or . . .

> <ns:deleteResourceResponse>
> <ns:return>
> **errorDescription**
> </ns:return>
> </ns:deleteResourceResponse>

. . . if something went wrong, where "errorDescription" is the description of the error occurred (typically the trace of the thrown exception).

3.1.3 update

You can invoke method `update` as follows

> serviceURL/update?resourceId=**resourceId**&repositoryId=**repositoryId**
> &newSecurityLevel=**newSecurityLevel**

where

resourceId is the id of the resource whose security level must be updated

repositoryId is the id of the repository where the resource is stored

newSecurityLevel represents the new security level of the resource. Can be either 0 (private) or 1 (public) or 2 (only for logged-in users) or 3 (only for members of the same group)

The result will be . . .

<ns:updateResponse>
<ns:return/>
</ns:updateResponse>

...if everything went fine or ...

<ns:updateResponse>
<ns:return>
errorDescription
</ns:return>
</ns:updateResponse>

...if something went wrong, where "errorDescription" is the description of the error occurred (typically the trace of the thrown exception).

3.1.4 exists

You can invoke method `exists` as follows

serviceURL/exists?resourceId=**resourceId**&repositoryId=**repositoryId**

where

resourceId is the id of a resource

repositoryId is the id of the repository where the resource is supposed to be stored

The result will be ...

<ns:existsResponse>
<ns:return>
'0', '**answer**'
</ns:return>
</ns:existsResponse>

...if everything went fine or ...

<ns:addResourceResponse>
<ns:return>
'1', '**errorDescription**'
</ns:return>
</ns:addResourceResponse>

...if something went wrong.

- In the first case "answer" is either `true` or `false`
- In the second one "errorDescription" is the description of the error occurred (typically the trace of the thrown exception)

3.1.5 getSecurityLevel

You can invoke method `getSecurityLevel` as follows

> serviceURL/getSecurityLevel?resourceId=**resourceId**&repositoryId=
> **repositoryId**

where

resourceId is the id of a resource whose security level must be retrieved

repositoryId is the id of the repository where the resource is stored

The result will be ...

> <ns:getSecurityLevelResponse>
> <ns:return>
> '0', '**securityLevel**'
> </ns:return>
> </ns:getSecurityLevelResponse>

... if everything went fine or ...

> <ns:getSecurityLevelResponse>
> <ns:return>
> '1', '**errorDescription**'
> </ns:return>
> </ns:getSecurityLevelResponse>

... if something went wrong.

- In the first case "securityLevel" is either 0 (private) or 1 (public) or 2 (only for logged-in users) or 3 (only for members of the same group)
- In the second one "errorDescription" is the description of the error occurred (typically the trace of the thrown exception)

3.1.6 getResources

You can invoke method `getResources` as follows

> serviceURL/getResources?securityLevel=**securityLevel**

where **securityLevel** represents the security level of the resource and can be either 0 (private) or 1 (public) or 2 (only for logged-in users) or 3 (only for members of the same group). The result will be ...

> <ns:getResourcesResponse>
> <ns:return>
> '0', '**resources**'
> </ns:return>
> </ns:getResourcesResponse>

. . . if everything went fine or . . .

> <ns:getResourcesResponse>
> <ns:return>
> ‘1’, ‘**errorDescription**’
> </ns:return>
> </ns:getResourcesResponse>

. . . if something went wrong.

- In the first case “resources” is a comma-separated list of single-quoted (resourceId, repositoryId) pairs, each of which has the format

  `\‘resourceId\’, \‘repositoryId\’`

- In the second one “errorDescription” is the description of the error occurred (typically the trace of the thrown exception)

3.1.7 selectAllowedResources

You can invoke method `selectAllowedResources` as follows

> serviceURL/selectAllowedResources?resourceIds=**resourceIds**
> &repositoryId=**repositoryId**&userId=**userId**

where

resourceIds is a comma-separated list of single-quoted ids of the resources a user is trying to access

repositoryId is the id of the repository where the resources are stored

userId is the id of the user trying to access the resources

The result will be . . .

> <ns:selectAllowedResourcesResponse>
> <ns:return>
> ‘0’, ‘**resources**’
> </ns:return>
> </ns:selectAllowedResourcesResponse>

. . . if everything went fine or . . .

> <ns:selectAllowedResourcesResponse>
> <ns:return>
> ‘1’, ‘**errorDescription**’
> </ns:return>
> </ns:selectAllowedResourcesResponse>

. . . if something went wrong.

- In the first case “resources” is a comma-separated list of single-quoted resource ids
- In the second one “errorDescription” is the description of the error occurred (typically the trace of the thrown exception)

3.1.8 commit

You can invoke method `commit` as follows

serviceURL/commit

The result will be . . .

```
<ns:addResourceResponse>
<ns:return>
</ns:return>
</ns:addResourceResponse>
```

. . . if everything went fine or . . .

<ns:addResourceResponse>
<ns:return>
errorDescription
</ns:return>
</ns:addResourceResponse>

. . . if something went wrong, where “errorDescription” is the description of the error occurred (typically the trace of the thrown exception).