FEARS Access Control Specifications

Gonçalo Dumiense

INESC-ID/IST
Software Engineering Group
Rua Alves Redol Nº 9, 1000-029 Lisboa
dumiense@rnl.ist.utl.pt

Abstract.

1 Access Control Requisites and assumptions

1.1 Roles

It is assumed that there are the following instances of access control roles:

- RegisteredUser
- SuperUser
- FearsAdministrator

It is assumed that roles are hierarchical, i.e., a FearsAdministrator is a RegisteredUser, a SuperUser is a FearsAdministrator, and consequently, a RegisteredUser.

1.2 Rules

The following rules should be enforced:

- Only registered users can vote, add features, and add comments.
- A registered user can remove votes, but only his own votes.
- Only Fears administrators can create and delete projects.
- Only a Super User can nominee a registered user as a Fears administrator, or dispromote him.
- Seeing existing projects and features requests is public.

2 Specification in DMAPL

2.1 Only registered users can vote, add features, and add comments

The following DMAPL code enforces the desired constraint:

```
// Only registered users can vote, add features and comments

RegisteredUserAccess :
  allow role RegisteredUser
  to @RegisteredUser
```

The annotation `@RegisteredUser` should be created:

```
package eu.ist.fears.server.domain.annotations

@Retention(RUNTIME)
@Target({METHOD, CONSTRUCTOR})
public @interface RegisteredUser {

  ...

}
```

The following methods should be annotated with `@RegisteredUser`

```
package eu.ist.fears.server.domain

class Project {

  ...

  @RegisteredUser
  public  void addFeature(FeatureRequest s){
    ...
  }

}

class FeatureRequest {

  ...

  @RegisteredUser
  public void vote(Voter voter) {
    ...
  }

  @RegisteredUser
  public void addComment(String comment, Voter voter) {
```

```
    ...
  }

}
```

2.2 A registered user can remove votes, but only his own votes

The following DMAPL code enforces the desired constraint:

```
// A registered user can remove votes, but only his own votes

RemoveVoteAccess :
  allow role RegisteredUser
  to eu.ist.fears.server.domain.FeatureRequest.
                                      removeVote(Voter voter)
  where { voter.equals(user);  }
```

2.3 Only Fears administrators can create and delete projects

The following DMAPL code enforces the desired constraint:

```
// Only Fears administrators can create projects

CreateProjectAccess :
  allow role FearsAdministrator
  to eu.ist.fears.server.domain.FearsApp.addProject(Project p,
                                                    Voter voter)

// Only Fears administrator can delete projects
DeleteProjectAccess:
  allow role FearsAdministrator
  to eu.ist.fears.server.domain.FearsApp.
                                      deleteProject(String name)
```

Alternativelly, one can create an annotation `@FearsAdministratorTask`, as decribed in Section 2.1 and simply write the following rule:

```
// Only Fears administrators can do Fear Administration Tasks

FearAdministrationAccess :
   allow role FearsAdministrator
   to @FearsAdministratorTask
```

The methods `addProject(Project p, Voter voter)` and `deleteProject(String name)` of class `FearsApp` should be annotated with annotation `@FearsAdministratorTask`

2.4 Only a Super User can nominee a registered user as a Fears administrator or dispromote him

The following DMAPL code enforces the desired constraint:

```
// Only a Super User can nominee a registered user as a
// Fears administrator

AddAdminAccess :
  allow role SuperUser
  to eu.ist.fears.server.domain.addAdmin(Voter v)

//Only a Super User can dispromote a FearsAdministrator

RemoveAdminAccess:
  allow role SuperUser
  to eu.ist.fears.server.domain.removeAdmin(Voter v)
```

As in Section 2.3 , one can use annotations to write just one simple rule:

```
SuperUserAccess:
  allow role SuperUser
  to @SuperUserTask
```

The annotations creation and method annotation is similar to what was refered in Section 2.3.

A third method possible approach for this specific rule is to admit that SuperUser does not correspond to a role, but to a specific user. Let us consider that, that specific user, is identified by the string `root`. Then the following code expresses this modified constraint:

```
SuperUserAccess:
  allow user root
  to @SuperUserTask
```