ACL

# Operational Concept Description

## Vision

The ACL application provides the ability to simplify the role based authorization and object instance based authorization. Use minimal technical concept to reduce the authorization complexity. Make the authorization module can be pluggable and decouple the authorization data from business data.

### Stakeholders

|  |  |  |  |
| --- | --- | --- | --- |
| **Stakeholder** | **Representative** | **Role** | **Objective** |
| Product | James Fu | Application Architect | * Leverage existing design and process for ACL * Increase diagnostic capabilities to determine root causes of problems |

## System Objectives

The ACL application provides a means for:

1. Unify the static resource role based protection with object instance based authorization mechanism. Make the authorization framework self are easy for testing.
2. Decouple the authorization data from business data and get rid of extra steps for ACL population. Isolate the ACL impact for multiple business modules.
3. Provide pluggable authorization ACL interceptor into the platform.

### Business Workflow

For a given system, the authorization would happen on following kind of resources:

1. The system function entry point such as: the restful URI address, server side Java API.
2. The object instance level access control such as: given user can read project 1 but cannot read project 2.
3. Organization hierarchy access control rule definition such as: all projects in SFO office can be read by SFO office manager.

All those kinds of resources can be defined as protected resources, for each resource we can assign unified resource id for them, based on the resource id we can grant specific role to given member (member could be a names user id, it also could be a group id). With that, we can define below structure:

acl\_objects\_members\_roles:

|  |  |  |
| --- | --- | --- |
| protected\_resource\_id | member | role |
| / | Administrator | /Administrator |
| /UI/ | UI.Administrator | /User |
| /UI/admin | UI.Admin.Administrator | /User |
| /UI/organization | UI.Administrator | /User |
| /UI/project | UI.Project.Administrator | /User |
| /project.Project/ | Project.Administrator | /project.Project/Administrator |
| /project.Project/ | Project.Viewer | /project.Project/Viewer |
| /project.Project/1 | Project.1.Administrator | /project.Project/Administrator |
| /project.Project/1 | Project.1.Viewer | /project.Project/Viewer |
| /project.Project/2 | Project.2.Administrator | /project.Project/Administrator |
| /project.Project/2 | Project.2.Viewer | /project.Project/Viewer |
| /structure.Organization/ | Organization.Administrator | /structure.Organization/Administrator |
| /structure.Organization/1 | Organization.1.Administrator | /structure.Organization/Administrator |
| /structure.Organization/2 | Organization.2.Administrator | /structure.Organization/Administrator |

* protected\_resource\_id: this field would follow the pattern </moduleName.typeName/id>, e.g.: resource / stands for the whole system; /project.Project/ stands for the whole project module Project type; /project.Project/1 stands for the specified project whose id is 1;
* member: this field could be the real user id or the group id which contains given user.
* role: this field stands the given member have what kind of role on it, later, the role can be mapped to operation which the member can perform on the given protected resource.
* role inheritance: the role on parent level resource would be inherited by sub resources, e.g.: member <Project.Viewer> has role </project.Project/Viewer> on resource </project.Project>, this attribute will be inherited to resource </project.Project/1> and </project.Project/2>, because </project.Project/1> and </project.Project/1> inherited from </project.Project/>
* The role based access control can be supported by acl\_objects\_members\_roles with following SQL(based on mySQL):

SELECT

DISTINCT(all\_resources.protected\_resource\_id),

CONCAT(

'member <'

, users\_resources.member

, '> can perform operation on <'

, all\_resources.protected\_resource\_id

, '> by role <'

, users\_resources.role

, '>'

) member\_permission

FROM

acl\_objects\_members\_roles all\_resources

, acl\_objects\_members\_roles users\_resources

WHERE

all\_resources.protected\_resource\_id LIKE '/UI/%'

AND all\_resources.protected\_resource\_id LIKE CONCAT( users\_resources.protected\_resource\_id, '%')

AND users\_resources.member='UI.Administrator'

-- this parameter will be specified for given member

ORDER BY users\_resources.member ,all\_resources.protected\_resource\_id ,users\_resources.role

Once the member role defined, we need specify what kind of operations can be performed by the given role, defined the below structure:

acl\_roles\_operatives

|  |  |  |
| --- | --- | --- |
| **role** | **operative** | **description** |
| /Administrator | / | administrator for all |
| /project.Project/Administrator | /project.Project/ | administrator for all granted projects |
| /structure.Organization/Administrator | /project.Project/ | organization owner can perform administrator actions for all projects belongs to the orgs |
| /project.Project/Viewer | /project.Project/view | viewer for all granted projects |
| /structure.Organization/Administrator | /structure.Organization/ | administrator for all granted orgnaizations |

* operative: this field follow pattern </moduleName.typeName/action>,e.g.: </> stands for all operation; </project.Project/> stands for all operation on module project type Project; </project.Project/view> stands for view operation on module project type Project;
* operative inheritance: the operative for given role would be inherited by sub operatives, e.g.: role </project.Project/Administrator> can perform operative </project.Project/> it also can perform operative </project.Project/view> because operative </project.Project/view> inherited from operative </project.Project/>; role</Administrator> can perform operative </> it also can perform operation </project.Project/> and </project.Project/view> because both </project.Project/> and </project.Project/view> inherited from operative </>

So far, we have protected resource id, member, role and operative, the remained part would be how the business data associated with the protected resource id, following table demonstrated the approach:

acl\_resource\_ids

|  |  |  |  |
| --- | --- | --- | --- |
| **module** | **type** | **id** | **acl\_resource\_id** |
| project | Project | 1 | /project.Project/1 |
| project | Project | 1 | /structure.Organization/1 |
| project | Project | 2 | /project.Project/2 |
| project | Project | 2 | /structure.Organization/1 |
| project | Project | 3 | /project.Project/3 |
| project | Project | 3 | /structure.Organization/2 |
| project | Project | 4 | /project.Project/4 |
| project | Project | 4 | /structure.Organization/2 |
| structure | Organization | 1 | /structure.Organization/1 |
| structure | Organization | 2 | /structure.Organization/2 |

* module: …….
* type: …….
* id: given module/type record original id.
* acl\_resource\_id: the generated protected resource id which will be used in the authorization system.
* associate business data with organization structure hierarchy: sometimes, the access permission for given business data such as: project are granted on organization hierarchy perspective, so, we need a approach to support that; the most popular scenario is: the business data self would have obviously link with the organization hierarchy such as: project would have the field <owner> which point to the Organization type. Let’s say project 1 belong to same Organization <1>, to simplify the permission grant on Organization level, during project 1 creation phase would generate 2 ACL resources ids they are:</project.Project/1> and </structure.Organization/1>; Since the Organization self is hierarchy, if Organization 1 have parent organization such as Organization root, one more resource id would be created </structure.Organization/root>. These kind of behavior could be happened during the project creation phase, for given project, it’s hierarchic organization information is down to top one way cascade, so, the cost on calculate all the parent organization would be cheap.
* following SQL(based on mySQL) demonstrated how to check whether given member can view given project:

*SELECT*

*CONCAT(*

*'member <'*

*, omr.member*

*, '> can perform operation<'*

*, ro.operative*

*, '> on <'*

*, ari.id, '[acl\_resource\_id=',ari.acl\_resource\_id ,']'*

*, '> by role <'*

*, ro.role*

*, '>'*

*) member\_permission*

*FROM*

*acl\_objects\_members\_roles omr*

*, acl\_roles\_operatives ro*

*, acl\_resource\_ids ari*

*WHERE*

*omr.role = ro.role*

*AND ari.module = 'project'*

*AND ari.type = 'Project'*

*AND ari.id = '1'*

*-- this parameter will be specified for given project*

*AND omr.member = 'Organization.Administrator'*

*-- this parameter will be specified for given member*

*AND ro.operative IN ( '/' , '/project.Project/' , '/project.Project/view' )*

*-- this parameter will be specified when the user action can be determined e.g.: /Project/view*

*-- once it finalized, need splited by '/' e.g.: / , /project.Project/view , /project.Project/*

*AND ari.acl\_resource\_id LIKE CONCAT(omr.protected\_resource\_id,'%')*

*-- TODO: this need be optimized*

*ORDER BY*

*omr.member , omr.protected\_resource\_id , ro.operative*

### Appendix

MySQL based schema for ACL

