A New Access Control Model for Manufacturing Grid

Zhihui Ge, Taoshen Li
School of computer science, Electronics and Information, Guangxi University, Nanning, China, 530004
Email: gezhihui@foxmail.com

***Abstract*—In order to protect the sensitive information in collaborative manufacturing grid environment, an access control solution was proposed to satisfy the inherent dynamic natures of the Manufacturing Grid, including dynamic Business Flow and system environment. Activity is introduced to encapsulate role and permission. Activity state, activity hierarchy and activity dependence are used to provide dynamic authorization and flexible multi-granularity permission management, which can get adapted to the dynamic, flexible modern business process. UNIX-like permission can guarantee default minimum read/write/delete permissions. This proposed model can meet the need of MG.**

***Index Terms*—manufacturing grid, access control, RBAC, task context, environment context**

I. Introduction

Manufacturing grid is such kind of platform, which can integrate various resources to form a giant “virtual computer”, provided for modern enterprise. In this “virtual organization”(VO), users can share resources and cooperate with problem solving. So, manufacturing grid has not only the same features as traditional network, but also has high dynamics, accurate organization, large scale etc.

As a mass customized distributed cooperation system, there are lots of data, which may come from different users, departments, enterprises, to be processed in manufacturing grid. And all the data are associated with fund, cost, product, project process and staff management etc privacy information. So how to guarantee the security of those data is very challenging. As an important component of security, access control can prevent data suffering from illegal modification and damage.

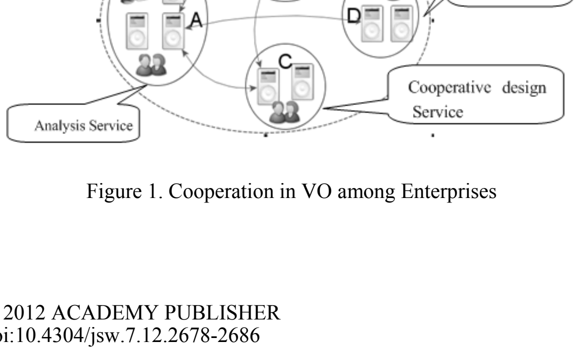

Figure 1. Cooperation in VO among Enterprises

Figure 1 shows a typical scenario, within which multiple manufacturing enterprises cooperate with each other based on virtual organization.

We assume enterprise A has found some opportunities in the market. In order to response quickly, A authorize B and C to complete the design of components and the final product integration. When the design work is completed, A delegates the manufacturing task to D. During this commercial activity, multiple enterprises associate and cooperate with each other through resources sharing and division of labor. Different services provide by various enterprises form a temporary virtual organization, once the manufacturing activity ends, the VO will naturally dissolved.

During the phases of cooperation, in order to ensure the final product satisfied with the primitive requirement, The staff participated in the collaborative design work have to communicate with each other, and the manager also needs to track and control the project process, coordinates developers and resources. All the tasks mentioned above need mutual cross-domain access. However, each enterprise joined in the VO has their individual security requirements and access control policy.

In this paper, a context-aware access control model is proposed, which can effectively support the interoperability with dynamic changing right polices for collaboration in manufacturing grid environment.

II. Related Work

Researchers have done many works in the field of grid. Grid Security Infrastructure (GSI)[3] is the essential middleware for authentication in grid environment. GSI maps the global user who needs to access resources to an account on local resource servers. Because the giant amount resources and users in grid environment, the mapping table will also be huge, furthermore GSI does not have effective global/local permission management scheme. Ian Foster etc proposed the Community Authorization Service (CAS)[4]. CAS allows resource providers to delegate some of the authority for maintaining fine-grained access control policies to communities, while still maintaining ultimate control over their resources. In order to gain access to a CAS-managed community resource, a user must first acquire a capability from the CAS server. So the final permission assigned to user is an intersection of VO (Virtual Organization) and resource provider. But CAS is static delegation of authority, which can not satisfy the requirement of dynamic authorization.

There are lots of research works on extension of traditional RBAC in grid environment. The model proposed in Ref[5] can provide dynamic permission according to gird environment context, but drawback is the dynamic changes of task in manufacturing project are not considered.

Recent years, along with the rapid development of manufacturing grid, some access control models are successively proposed. In order to support interaction between global and local, dynamic and static security strategies in dynamic heterogeneous manufacturing grid, Ref[6] proposes an extension method for RBAC. The model is based on CAS, but it is too complicated and can not effectively reflect the changes of environment context so as to control view of CAD model.

III. Group-Activity Based Access Control Model

A. Design Philosophy

The access control model we designed is showed in Figure 2, which is an activity-centered, encapsulated with role and right. The work flow can easily be organized and coordinated to form a global management and dynamic authorization. The group provided autonomous management and UNIX-like permission configuration all can improve the efficiency of administrator.

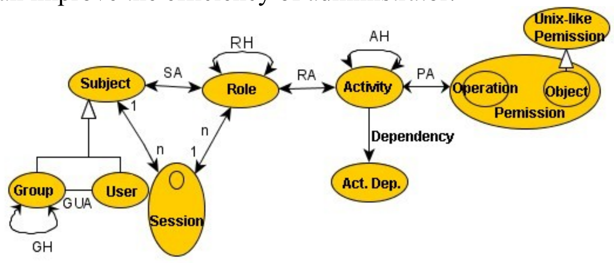

Figure 2. GA_RBAC Access Control Model

B. Definitions

Definition 1. ***Object and UNIX-like permission object base class:*** The most important part in access control system is to determine the object to be controlled. But all data and resources are dynamically increasing in manufacturing grid, and most part of them are stored in database. It's very hard for the administrator to assign all the permisssions. So we define the following UNIX-like configuration for objects.

$$Permission = \{\langle obj\_id, owner\_id, owner\_group, unix\_perms \rangle\}$$

obj_id and *owner_id* are object and owner identity respectively. *unix_perms* is 32 bit integer, which records the read/write/delete access rights. A shared object may be composed of multiple objects and organized in hierarchical structure. The products designed by different designers, who are affiliated with different enterprises, belong to different organizations. The *owner_group_set* is used for such situation, which can solve rights inheriting problem of sub-objects.

Definition 2. ***Subject:*** subject is an object, which is assigned some role and participated in collaborative activities with corresponding permission. Subject initiates resource request, it is a abstract concept, which can be person, program and even group.

Definition 3. ***Groups and Group Hierarchy (GH):*** one enterprise is composed of multiple departments which are associated with different roles. The organizations in VO may be enterprises and departs coming from different domains, so we give the following definition to describe this relation.

$$G = \{\langle gid, super\_gid, u\_set, r\_set \rangle\}$$

gid is group identity; *super_gid* is parent group of *gid*; *u_set* and *r_set* are user set and corresponding role set of group respectively.

Definition 4. ***User:*** user is a staff with some role and taking part in some activity.

$$User = \{\langle uid, group\_set, r\_set \rangle\}$$

uid is user identity, *group_set* is the user's group; *r_set* is role set assigned to user.

Definition 5. ***Group-User Relation (GUA):*** *GUA* indicates the many-to-one relation between user and group set.

$$user: \{\exists gh \in GH, \exists! g \in gh, \langle g, u \rangle\}$$

Definition 6. ***Role and Role Hierarchy (RH):*** *Role* is the entity who owns rights, which is related to task function, responsibility etc semantics.

$$Roles = \{\langle rid, super\_rset, group\_set, u\_set, actset \rangle\}$$

rid is the role identity, *super_rset* is the role set inherited. *group_set* and *u_set* are group and user set respectively. *actset* is activity set the role participated in.

In manufacturing grid, role can be system role such as administrator, service provider, service caller etc, and it also can be set according to the specific task and position, such as design engineer, process engineer, design leader etc.

Definition 7. ***Permission:*** *permission* is the permitted operations on specific object.

$$Perms = \{\langle op, obj \rangle\}$$

Definition 8. ***Activity and Activity Hierarchy (AH):*** *activity* is the base unit of decomposed task and their hierarchy relation, which also is encapsulated with role and permission etc objects.

$$Acts = \{\langle aid, state, superAct, subActs, rset, peSet \rangle\}$$

aid is the activity identity, *state* is the activity state, *superAct* is the father activity, *subActs* is the child activity set, *rset* is the role set of participators, *peSet* is the record set of right.

During the cooperation process of VO, some project cycle may be divided as several activities. Furthermore, activity can be divided into tasks, so the activity hierarchy structure can express this relation well.

Definition 9. ***Subject-Role Relation (SR):*** *SR* is a kind of many-to-many relation.

$$SR = \{\langle sid, rid \rangle\}$$

sid can be group id or user id, *rid* is role identity.

Definition 10. ***Role-Activity Relation (RA):*** *RA* is many-to-many relation between role and activity.

$$RA = \{\langle rid, aid \rangle\}$$

rid is role identity, *aid* is activity identity.

Definition 11. ***Activity-Permission Relation (AP):*** *AP* is many-to-many relation between activity and right.

$$AP = \{\langle aid, peSet\rangle\}$$

Activity A can be divided into seven sub-activities and they have correlations with each other, figure 3 shows the activity dependence graph among them.

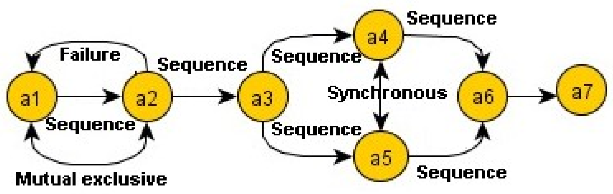

Figure 3. Activity Dependence Graph

Definition 12: ***Activity State Migration (ASM)*** is the migration of activity state, which is used to describe the changes of activity state.

$$AS = \{\langle all, inactive, active, denied, complete\rangle\}$$

all is used to guarantee some right can be available at any time, such as the right management of administrator. *active* and *inactive* indicate the activity is in its active or inactive state respectively. *denied* state is used to describe backtracking of activity dependence. *complete* indicates the activity is completed, all the rights are not available except the read right of its owner. The migration of activity state is showed as figure 4.

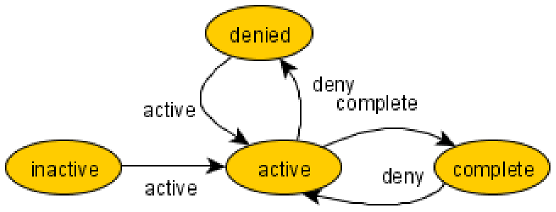

Figure 4. Activity State Migration

Definition 13: ***Activity Dependence (AD)*** is the relationship of activities, which includes:

(1) **Mutually Exclusive Dependence**: the exclusive dependence of different activities when executing. For any two activities a_i and a_j, a_j can not enter into that state when a_i has entered, vice versa. Noted as a_i(state) $\leftrightarrow \neg$ a_j(state) or a_j(state)$\leftrightarrow \neg$ a_i(state). Mutually exclusive dependence meets the conditions of non-reflexivity, non-transitivity and symmetry.

(2) **Sequence Dependence**: the precedence relationship of different activity when executing. For any two activities a_i and a_j, when a_i enters into complete state, the active state of a_j is activated. Noted as a_i (*complete*)$\rightarrow$$a_j$ (*active*).

(3) **Failure Dependence**: the denied relation of different activities when executing. For any two activities a_i and a_j, when a_i enters into denied state, the active state of a_j is activated. Noted as a_i (*denied*)$\rightarrow$$a_j$ (*active*).

(4) **Synchronous Dependence**: synchronous execution of different service state. For any two activities a_i and a_j, when a_i is implemented, a_j has to be implemented at the same time, vice versa. We note this as a_i (state)$\leftrightarrow$ a_j (state). Synchronous relation is a kind of equivalence relation, which meets the conditions of reflexivity, transitivity, symmetry.

Definition 14: ***Activity Property (AP)***: we define activity A has the following properties.

(1) **Start Activity Set**: in activity hierarchy and activity dependence, $StartAct$ is composed of all the started sub-activities, which can be formalized as

$$StartAct(A) = \{a_i | \forall a_i \in A(\forall a_j \in A)[a_i \neq a_j \rightarrow \langle a_j, a_i\rangle \neq Sequence]\}$$

It means that the start activity set of activity A is the activity set of whose sequence dependence in-degree is zero.

(2) **End Activity:** In order to guarantee the cooperation going successfully, we set each activity having only one ending sub-activity $EndAct(A) \subseteq$ is the activity set of whose sequence dependence out-degree is zero.

$$EndAct(A) = \{a_n | \exists! a_n \in A(\forall a_j \in A)[a_n \neq a_j \wedge \langle a_n, a_j\rangle \neq Sequence]\}$$

(3) **Activate Activity**: When some activity instance a_i is running, all the conditions are satisfied, then a_i is activated and all its permission set are available.

$$activate(a_i) = \cup\, deptype(a_i, a_j) \quad a_i, a_j \in A, i \neq j$$

deptype is the activity dependence type.

(4) **Prior Activity Set**： For the convenient of managing the activation of activity, we need to find out the prior activity set of some activity.

$$PriorAct(a_i, deptype) = \{a_p | \forall a_p \in Acts \wedge deptype(a_p, a_i) \wedge a_p \neq a_i\}$$

(5) **Follow-up Activity Set**：We also need to find out the follow-up activity set of some activity.

$$FollowAct(a_i, deptype) = \{a_f | \forall a_f \in Acts \wedge deptype(a_i, a_f) \wedge a_f \neq a_i\}$$

IV. Model Operations

The implementation process of access control model is mainly include static permission assignment, dynamic activity management and user authorization.

A. Static Permission Assignment

Any implementation of access control model has to provide permission policies for the access control system. Static permission assignment is a method to achieve permission policy, which can initialize the data for access control system. In our model, the static permission assignment includes activity partition, activity permission assignment, activity role association, subject role association.

The activity partition is the first step. The administrative staffs organize and partition the activities at different stages during the work flow and decide their dependence relationship. This task is related to special application. When the activity and activity dependence is worked out, permission can be assigned by *RA* and *AP* mapping, and then mapping the available roles for users and their groups according to the *SR*.

B. Activity Management and Dynamic Permission Adjustment

Activity management is one key component to achieve the goal of dynamically adjust the permission of users. According to the features of access control model of manufacturing grid, we design three operations to complete the activity management.

(1) Startup() function is used to activate some activity and its corresponding sub-activities. In *AH*, when we want to activate an activity from its inactive state, we must keep searching for the startup activity set in the sub-activity set until there are no more sub-activities due to the hierarchy organization of *AH*. The operations can be constructed according to the definition of **Start Activity Set**, the pseudo code is as following:

```
Startup(act)
Step1: tmp_act=act，
          if tmp_act.state!=all
             tmp_act.state=active
Step2: if tmp_act.subActSet.length=0
             return
Step3: if tmp_act.subActSet.length!=0
             if tmp_act.SubActs_Dep!=null
                for each start_act in StartAct(tmp_act)
                   Startup(sub_act)
             else
                for each act in tmp_act.subActSet
                   Startup(sub_act)
```

Figure 5. Function for Starting up an Activity

(2) After some activity is completed or denied, the activities which having dependence with it have to be activated to proper state. ActivateAct() is used to dynamically adjust the state of activities. The main idea of this operation is to find a given activity a_i's *FollowAct(a_i,deptype)* set according to the state of activity a_i in *AD*. If activities in *FollowAct(a_i,deptype)* set is satisfied with the dependence relation and corresponding state, then it will be activated.

```
ActivateAct(a_i,state)
Step1: a_i.state：= state
Step2:  if state=denied
              followActs:=FollowAct(a_i,failure)
           else
              if state=complete
                 followActs=FollowAct(a_i,order)
Step3: for each tmp_act in followActs
              preFailureSet:=PriorAct(tmp_act,failure
)
              preOrderSet:=PriorAct(tmp_act,order)
         for each act in preFailureSet
                 if act.state=denied continue
                 else return
         for each act in preOrderSet
                 if act.state=complete
                    continue
```

Figure 6. Function for Activating an Activity

(3) CompleteAct() is used to mark the completion state of activity. In *AD*, when the complete sub activity of one activity is completed, that indicated the activity is also completed.

```
CompleteAct(act)
Step1: if act.state = all
          return
       else act.state=complete
              goto step2
Step2: tmp_act:=act.super_act
Step3: if tmp_act = null
          return
       else
          if EndAct(tmp_act)=act
             ActivateAct(act,Order)
             CompleteAct(tmp_acct)
          if EndAct(tmp_act)!=act
             return
```

Figure 7. Function for Completing an Activity

C. User Authorization

The object of access control is to judge the requests of users, and then decides whether the requested access can be implemented on the resource. So in our access control model, we introduce activity state, group and corresponding UNIX-like permission control. The request of user can be expressed as following

$$AccessRequest\langle u_{id}, group_{set}, act_{id}, op, obj\rangle$$

u_id is user identity, *group_set* is the group of user, *r* is the activity role of user in current session, *act_id* is the activity identity of current request of user, *op* is the requested operation, *obj* is the requested object. We design the following strategy:

(1) If the activity is in its inactive state, then all the rights are not available during all the activities.

(2) If activity is in its denied state, then reset the activity into active state.

(3) If activity is in active or all state, then all the rights are available.

(4) If activity is in completed state, then only read-only right is available for the corresponding users.

V. Implementation

A. Framework

In order to illustrate the effectiveness and security of our access control model, we construct a prototype based on Globus Toolkit 4. The framework of our prototype is showed in Figure 8.

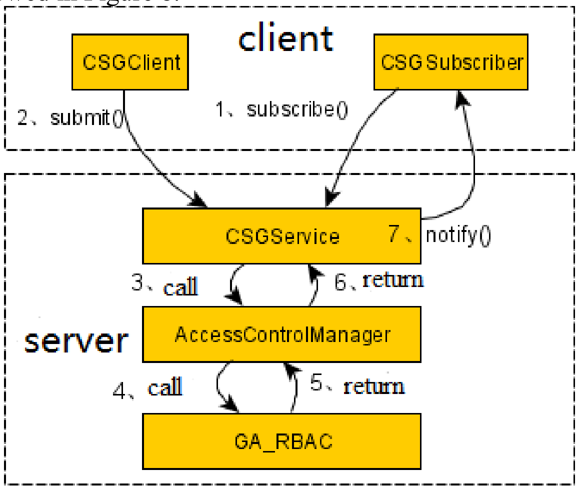

Figure 8. Framework of Prototype

(1) CSGService is a grid service based on Globus Toolkit 4, which is used to save the properties of resource objects.

(2) CSGClient is the client which is used to submit the users requests.

(3) CSGListener is the subscriber of CSG service, which is used to monitor the state changing of resources in CSGService.

(4) AccessControlManager is used to coordinate the interaction between CSGService and GA_RBAC.

(5) GA_RBAC is our proposed group-activity based access control model, which is used to authorize the users' requests.

B. WSDL based Service Description

We use WSDL (Web Service Description Language) to describe the interaction protocol between server and client.

(1) Submit user's request. The cooperation works among clients are implemented by invoking the grid service of server. We define the message format of SubmitRequest operation as following

<SubmitRequest> : = AccessRequest

<AccessRequest>:=<Use,ActID,Op,PermissionBaseObject>

<User>:=<u_id,u_name,u_ip,u_i,time,groupmembership>

<ActID>:= Identification of current user's activity

<Op>:= add | create | modify | delete | save | load | read | write | read …

<PermissionBaseObject>:=<ObjID,ObjName,ObjOwner,ObjOwnerGroup,ObjUnixPerms >

The most important part of our prototype is the access control, so we add all the items needed by authorization in SubmitRequest. User indicates the sender of the request, in which u_id is the user's identification, u_name is the name of user, u_ip is the IP address, time is the request sending time, groupmembership is the user's group. ActID is the id of current activity, Op is the operation of user request. PermissionBaseObjec is the base class of operation object. The object of our prototype simulates the permissionmanagement system of UNIX to simplify the configuration of permission, so we abstract the basic permissionclass of UNIX as PermissionBaseObject.

(2) Submit user's response. After the user submits request, the server will send feedback to clients according to the user's request. The user's request accepted, the server will update the user's UI as the correct result is sent back to the client to assure the consistency of the client and the server. Otherwise, the server must notify the client if the user's request is not accepted. So we define the message format of SubmitResponse as following

SubmitResponse:=<StatusCode, PermissionBaseObject>

StatusCode indicates the completed state code of operation, a Boolean value. PermissionBaseObject is the permission object, which is the XML date sent back by the server to the client.

(3) Notification Mechanism. The notification mechanism is achieved by using the features of grid service that can maintain the state of objects the users operates. When the state of some resources changes in grid service, the client monitoring the resource can be notified immediately. We design an interface CSGServicePortType inherited from GetMultipleResourceProperties, NotificationProducer which are defined in WSRF specification. The WSDL file is showed in Figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CSGService"
targetNamespace="http://www.csg.net/namespaces/csg/CSGService"
  xmlns="http://schemas.xmlsoap.org/wsdl/"……>
<types>
 <xsd:complexType name="PermissionBaseObject"
abstract="true">
                <xsd:sequence><xsd:element
```

```
ref="tns:ID"/>        ……
                          <xsd:any minOccurs="0"/>
                          </xsd:sequence>
  </xsd:complexType>
 <xsd:complexType name="CSGSolid" abstract="true">
….
</xsd:complexType>……
<xsd:element name="PermissionObject" type="tns:
PermissionBaseObject"/>……>
            <!-- RESOURCE PROPERTIES -->
            <xsd:element name="CSGSolidsProperties">
            <xsd:complexType>
                        <xsd:sequence>
                                    <xsd:element
ref="tns:SubmitRequest" minOccurs="1"
maxOccurs="1"/>
                </xsd:sequence>
            </xsd:complexType>
            </xsd:element>
</xsd:schema>
</types>
<!--MESSAGES-->
<message name="SubmitInputMessage">
            <part name="parameters"
element="tns:SubmitRequest"/>
</message>
<message name="SubmitOutputMessage">
            <part name="parameters"
element="tns:SubmitResponse"/>
</message>
<!--P O R T T Y P E-->
<portType name="CSGServicePortType"
  wsdlpp:extends="wsrpw:GetResourceProperty
  wsrpw:SetResourceProperties
  wsrpw:GetMultipleResourceProperties
  wsrpw:QueryResourceProperties
  wsntw:NotificationProducer"
  wsrp:ResourceProperties="tns:CSGSolidsProperties">
            <operation name="submit">
                        <input
message="tns:SubmitInputMessage"/>
                        <output
message="tns:SubmitOutputMessage"/>
            </operation>
</portType>
</definitions>
```

Figure 9. Part of WSDL File

C. *GA_RBAC Implementation*

(1) Permission Basic Class

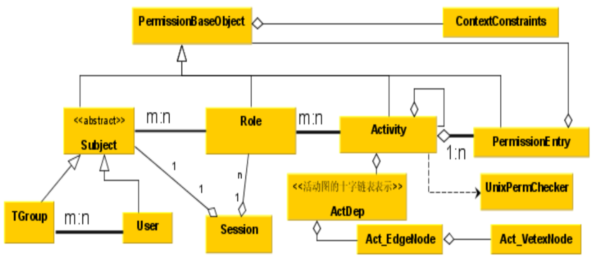

Figure 10. Class Diagram of GA _RBAC

Figure 10 is the class diagram of GA _RBAC model. In order to simplify the administrator's work, we provide the minimal permissionset for each object, which includes read/write/delete three operations. We abstract a permissionbasic class PermissionBaseObject for permission judgment showed in Figure 11.

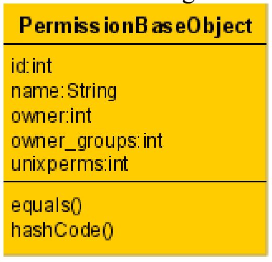

Figure 11. PermissionBaseObject Class

(2) Permission Record and PermissionChecker

PermissionEntry class is composed of by two-tuples <op, obj>, and it overrides the equals() and hashCode() functions to judge the permission record is equal or not. The class description is showed in Figure 12.

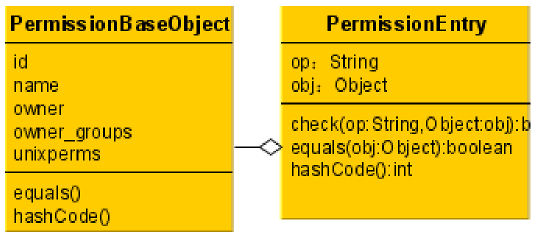

Figure 12. PermissionEntry Class

UnixPermChecker is used to check the permission of objects. The class description is showed in Figure 13.

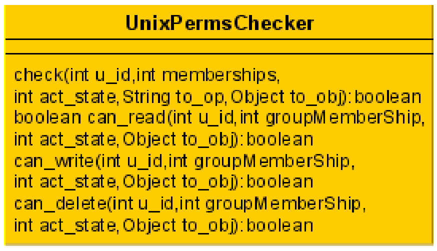
UnixPermsChecker
check(int u_id,int memberships,
int act_state,String to_op,Object to_obj):boolean
boolean can_read(int u_id,int groupMemberShip,
int act_state,Object to_obj):boolean
can_write(int u_id,int groupMemberShip,
int act_state,Object to_obj):boolean
can_delete(int u_id,int groupMemberShip,
int act_state,Object to_obj):boolean

Figure 13. UnixPermChecker Class

(3) Subject, Group and User Class

In manufacturing grid, the staffs in enterprise alliances and organizations are the subjects of collaboration. Usually, each enterprise has some predefined organization structure. For example, an enterprise has several departments, which is composed of several groups. There are several staffs in each group, who may also belong to different groups and departments. In order to present this kind of structure and realize authorization to original organization, the subject class is abstracted and the group adopts hierarchy structure. The user and group are multiple to multiple relation. The class description is showed in Figure 14.

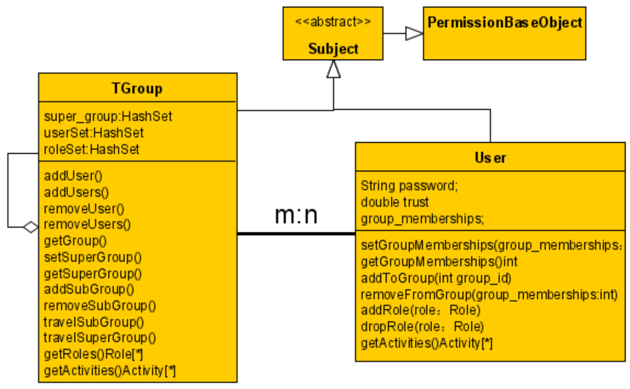

Figure 14. Subject, Group and User Class

(4) Role and Session

In GA_RBAC model, one role class indicates a role, which is associated with special subject and activity. In this way, the subject can operate according to the permission in the given activity. Session is mainly used to maintain the available roles for the logged users and operate according to the user's request. The class description is showed in Figure 15.

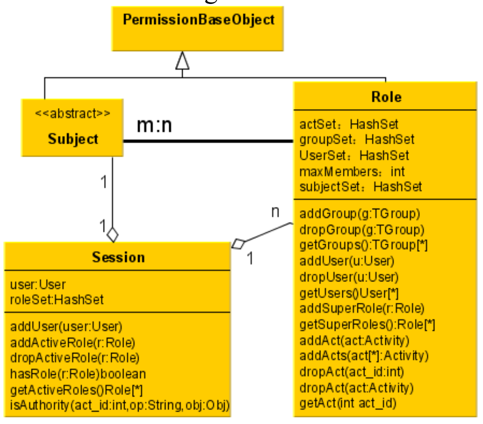

Figure 15. Role and Session Class

(5) Activity and Activity Dependence

Activity and Activity Dependence are two very important parts in GA_RBAC. They are the key to realize the dynamic permission adjustment. In our prototype, we use activity diagram to present activity dependence relation. Our activity centered authority judgment procedure is showed in Figure 17.

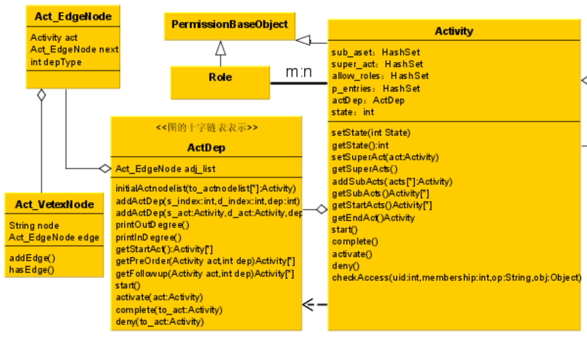

Figure 16. Activity and Activity Dependence Class

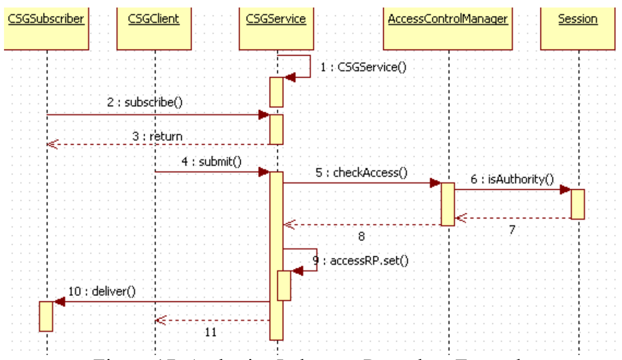

Figure 17. Authority Judgment Procedure Example

D. *Example*

We use the scenario showed in Figure 1 to illustrate our access model. There is a product design project CSGProject carried out by three groups: G_a, G_b and G_c. G_b and G_c are responsible for component design and integration design respectively. Project analyzer and project supervisor in G_a can provide requirement analysis and project tracking for G_b and Gc. We assume there are such users {u_{a1}, u_{a2}}, {u_{b1},u_{b2}}, {u_{c1}, u_{c2}}, the activities may include requirement analysis, analysis review, system design, component A design, component B design and integration design etc stages. The whole cooperation procedure is as following.

When CSGProject is started, the state of activity requirement analysis is transformed from inactive to active. At this time project analyzer can make requirement analysis and create, modify and edit the corresponding documents. When analyzer's work is done, his activity changes to state completed, the analysis review activity and the permission of monitor are all activated as the requirement documents submitted. If the analysis review cannot pass, then the state of it should be changed to denied, and make the requirement analysis activity which has failure dependence relation with it reenter into active state. Otherwise, the review activity becomes complete state, and system design is activated. Requirement analysis and analysis review are mutual exclusive dependence, so they must be carried out by different roles and users. The cooperation design will finally succeed. The activity dependence and tasks division is showed in Figure 18.

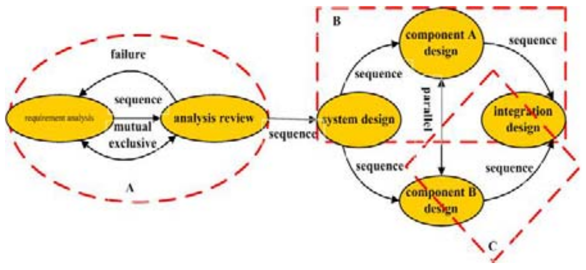

Figure 18. Activity Dependence

According static permission assignment rules, the related tables are as following.

TABLE 1
ACTIVITY OF CSGPROJECT

AH		Available Roles	Available Permissions
CSGPro	Analyse	Analyser	Create Document, Edit Document, default
	Verify	Verifier	Note Document, default
	GlobalDesign	Desinger	Create Document, Edit Document, default
	PaDesign	ADesigner	Edit Solid, default
	PbDesign	BDesigner	Edit Solid, default
	Assemble	Assembler	Create Solid, Boolean Solid, default
	admin	Admin	Manage Role, Manager Activity, default
	view	ProjectMember	Read

TABLE 2
ROLES HIERARCHY

Role Name	Parent Role
ProjectMember	
Analyser	ProjectMember
Verfier	ProjectMember
Designer	ProjectMember
PaMember	ProjectMember
PbMember	ProjectMember
Assembler	ProjectMember
Admin	ProjectMember

TABLE 3
GROUP HIERARCHY

Parent Group	Group
A	Analyser
A	SubProjectLeader(SubPL)
B	Designer
B	Draftman
B	SubProjectLeader(SubPL)
C	Draftman

TABLE 4
USER-GROUP MAP

User Name	Group
u_{a1}	A.Analyser
u_{a2}	A.SubProjectLeader(SubPL)
u_{b1}	B.Designer
u_{b1}	SubPL
u_{b2}	B.Draftman
u_{c1}	C.Draftman

TABLE 5
SUBJECT-ROLE MAP

Subject Name	Role
A	ProjectMember
B	PaMember
C	PbMember
A.Analyser	Analyser
A.SubPL	Admin
B.Designer	Designer
B.Draftman	PAMember, Assembler
C.Draftman	PbMember
C.Draftman	Assembler

VI. CONCLUSIONS

In traditional manufacturing grid access control model, it is hard to deal with the cooperation between dynamic and non-dynamic businesses and can not effectively support the global control and local autonomous management etc. In this paper, we propose a group-activity based access control model. Activity, activity hierarchy, activity state and activity dependence are introduced into our model, by this means, we can clearly describe the dependence of operations and authorize dynamically. The UNIX-like permission control make the right management is much easier.

ACKNOWLEDGMENT

This work is supported by Guangxi Nature Science Foundation (09-007-05S018, 2010GXNSFD013037), Guangxi Key Laboratory of Manufacturing System & Advanced Manufacturing Technology Foundation (11-031-12S02-02).

REFERENCES

[1] Fan Yushun. Concept and Architecture of Manufacturing Grid. Aeronautical Manufacturing Technology. 2005,l0,42-45.

[2] Foster I, Kesselman C and Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. International Journal of Supercomputer Applications, 2001,15 (3) : 200-222.

[3] I.Foster,C. Kesselman, G. Tsudik, S. Tuecke. A security architecture for computational grids. Proceedings of the Fifth ACM Conference on Computer and Communications Security, November 1998, pp. 83–92.

[4] Pearlman L, Welch V, Foster I. A community authorization service for Group collaboration. IEEE 3rd International Workshop on Policies for Distributed Systems and Networks. 2002, pp:50-59.

[5] YAO Han-Bing HU He-Ping LU Zheng-Ding LI Rui-Xuan. Dynamic Role and Context-Based Access Control for Grid Applications. PDCAT 2005, 404 - 406.

[6] CAI Hong-xia, YU Tao, FANG Ming-lun. Access control of manufacturing grid. Computer Integrated Manufacturing Systems. 2007,4(13) : 717-720.

[7] Hongxia Cai, Tao Yu, Minglun Fang. Access Control Model of Manufacturing Grid. IFIP International Federation for Information Processing, 2006, p. 938-943.

[8] F. Tao, L. Zhang, K. Lu & D. Zhao. Research on manufacturing grid resource service optimal-selection and composition framework. Enterprise Information Systems, 2012,Vol. 6, No. 2, 237-264.

[9] Dongming Zhao ; Yefa Hu ; Zude Zhou. Resource Service Composition and Its Optimal-Selection Based on Particle Swarm Optimization in Manufacturing Grid System. IEEE Transactions on Industrial Informatics, 2008,Vol. 4, No. 4, 315 – 327.

[10] Wenjun Xu, Zude Zhou, D. T. Pham, Quan Liu, C. Ji and Wei Meng. Quality of service in manufacturing networks: a service framework and its implementation. The International Journal of Advanced Manufacturing Technology. 2012,1-11.

[11] Zhengqiu He, Lifa Wu, Huabo Li, Haiguang Lai, Zheng Hong. Semantics-based Access Control Approach for Web Service. Journal of Computers, Vol 6, No 6 (2011), 1152-1161.

[12] Bailing Liu. Efficient Trust Negotiation based on Trust Evaluations and Adaptive Policies. Journal of Computers, Vol 6, No 2 (2011), 240-245.

[13] Xiaoming Wang, Yanchun Lin. An Efficient Access control scheme for Outsourced Data. Journal of Computers, Vol 7, No 4 (2012), 918-922.

Zhihui GE was born in Hebei, China, in 1978. He received a B.S. degree in Computer Science from the Beijing Technology and Business University, Beijing, China, in 2001, and a M.S. degree in Computer Science from the Guangxi University, Guangxi, China, in 2004 and a Ph.D. degree in Computer Science from the Central South University, Hunan, China, in 2007.

He is a associate professor in the Guangxi University. Nanning, Guangxi, China. His research interests are in networks and security with special emphasis on distributed system.