Front. Comput. Sci.
DOI 10.1007/s11704-016-5081-x

RESEARCH ARTICLE

Tenant-based access control model for multi-tenancy and sub-tenancy architecture in Software-as-a-Service

Qiong ZUO[1,2], Meiyi XIE (✉)[1], Guanqiu QI[2], Hong ZHU[1]

1 School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
2 School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe AZ 85287, USA

Abstract Software-as-a-Service (SaaS) introduces multi-tenancy architecture (MTA). Sub-tenancy architecture (STA), is an extension of MTA, allows tenants to offer services for subtenant developers to customize their applications in the SaaS infrastructure. In a STA system, tenants can create subtenants, and grant their resources (including private services and data) to their subtenants. The isolation and sharing relations between parent-child tenants, sibling tenants or two non-related tenants are more complicated than those between tenants in MTA. It is important to keep service components or data private, and at the same time, allow them to be shared, and support application customizations for tenants. To address this problem, this paper provides a formal definition of a new tenant-based access control model based on administrative role-based access control (ARBAC) for MTA and STA in service-oriented SaaS (called TMS-ARBAC). Autonomous areas (AA) and AA-tree are proposed to describe the autonomy of tenants, including their isolation and sharing relationships. Authorization operations on AA and different resource sharing strategies are defined to create and deploy the access control scheme in STA models. TMS-ARBAC model is applied to design a geographic e-Science platform.

Keywords Software-as-a-Service (SaaS), multi-tenancy architecture (MTA), sub-tenancy architecture (STA), role-based access control (RBAC) model, tenant-based access control model

Received February 27, 2015; accepted December 1, 2015

E-mail: qiong_zuo@hust.edu.cn; xiemeiyi@hust.edu.cn; guanqiuq@asu.edu; zhuhong@hust.edu.cn

1 Introduction

Cloud computing often has three principal components: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). Multi-tenancy architecture (MTA) is often used in SaaS where multiple tenants can use the same code base stored in the SaaS to develop applications. One tenant application may be under development, while the SaaS is executing other tenant applications at the same time.

A tenant can be a single user or an organizational entity. A tenant application can be used by many end users of the organization. Many organizations today have sub-organizations, e.g., a corporation can have multiple subsidiary companies, and these subsidiaries while different may share the significant requirements. For example, Wells Fargo Bank is an enterprise-level tenant of Salesforce.com. It has over 9 000 bank branches, about 270 000 employees, and serves 3 600 customers per branch[1)]. One branch may operate in the USA, while the other may operate in Asia, and these two are set up to meet local regulations, but both share the significant business operations. In this case, the company may be a tenant, while these two subsidiaries are subtenants. This is the sub-tenancy architecture (STA) [1] and this is an extension of MTA, and in STA, a tenant application can be further customized to form subtenant applications, and subtenants may share data and software with fellow subtenants or their

[1)] Wells Fargo's success story. http://www.salesforce.com/customers/stories/wells-fargo.jsp, 2013 parent tenants. Technically, a subtenant can also have its own sub-subtenants, however, the management of these sub-subtenants may be involved. Furthermore, a MTA may be considered as a sub-case for STA where no tenant has subtenants.

Before formally modeled as STA [1], some cloud service providers, such as Salesforce, NetSuite and OpenStack, use user groups and application subsets on MTA to practice hierarchical multi-tenants and their different application requirements. In MTA, each subsidiary of a tenant is built either as an independent tenant that is difficult for customized resource sharing or re-customization among relative tenants, or as a user of such tenant where the real world hierarchical relations, resource isolation and sharing relations are implied by complicated role assignment and constraints.

STA is a more flexible and extensible architecture for hierarchical multi-tenancy applications. In STA, multiple tenants co-exist. Each tenant should be autonomous and can authorize its subtenants to access its own resources, including private service components and data. Each subtenant may not only inherit its tenant's resources, but also customize its own applications and allow or forbid others to access its resources. Sibling-tenants may also share their service components or data with each other.

Currently, most existing access control models for MTA are based on role-based access control (RBAC) [2] or administrative role-based access control (ARBAC) [3]. These models can be divided into three categories: 1) using database schema and security strategies in data-centric clouds [4] without considering of service component sharing; 2) adding various kinds of hierarchy, constraints or management scope separation for multiple tenants isolation [5–10]; and 3) adding issuer-tenant federals for cross-tenant sharing of outsourcing components [11–13].

Besides the security strategies provided by MTA, STA access control needs to address the following issues:

1) Privacy sharing Tenants and their subtenants may share private service components and data. A tenant can grant its own resources including data and customized components to its subtenants, and meanwhile, may not allow its ancestor-tenants or system administrators to access to them, e.g., Fig. 1 shows that tenant T_1's private resource PR_1 is partially shared by its subtenant T_{11}. A tenant may also share its private components with its sibling, e.g., PR_1 is partially shared by its sibling tenant T_2, and T_{11}'s private resource PR_{11} is partially shared by its sibling subtenant T_{12}.

2) Autonomous tenants With respect to granting privileges to subtenants, tenants act like autonomous agents even though their operations are still confined by the SaaS infrastructure. A tenant manages its own resources, and can create its subtenants and grant its resource access privileges to them. A system administrator can create tenants and rent resources to them, but cannot interfere with tenants' internal affairs. Furthermore, due to privacy isolation, role privilege inheritance no longer exists in the system scope. What access privilege can/cannot be inherited and what resources can/cannot be cross-level controlled are different from traditional MTA systems. These need to be redefined.

3) Sharing relationships among tenants The sharing may be between two sibling tenants (e.g., T_1-to-T_2, or T_{11}-to-T_{12} in Fig. 1) or parent-child tenants (e.g., T_1-to-T_{11} in Fig. 1). Sharing resources include all kinds of resources, such as application components and data in service-oriented SaaS. Private resource components can be granted for other tenants to access with their owner's permissions. "Sharing directions" may be from a parent-tenant to its subtenants, or from a subtenant to its parent-tenant, or from a tenant to its sibling.

4) Shared components Components such as GUIs, workflows, services, and data components can be shared [14]. But different components with different access properties need to be differentiated. And a component may be a composite one, with sub-components coming from different providers or transmitted from different tenants. System administrator may not be the only resource owner. A tenant can be both a component producer and an application renter. The access control of "shared composite components" is complicated.

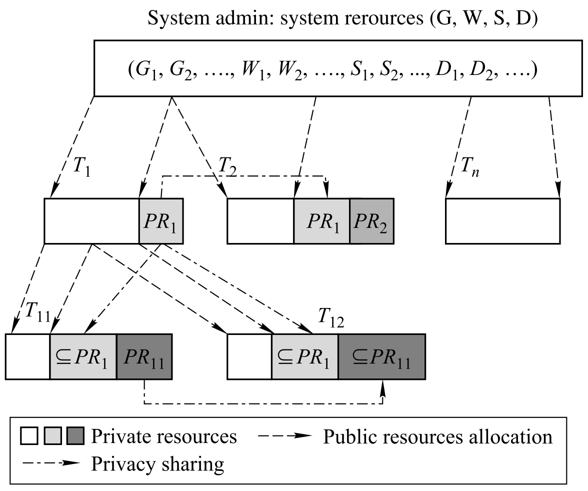


Fig. 1 System administrator, tenants and subtenants relationships

To address these problems, meanwhile concerning about the essential features for effective cloud authorization mechanism [15], a flexible, decentralized and scalable authentication and authorization model for different-level tenants to share existing resources, customize their own application and keep their secrets should be supported.

The contributions of this paper are as follows.

1) A new tenant-based access control model (called TMS-ARBAC) based on ARBAC for STA is presented. As STA includes MTA, this also applies to MTA.

2) Different resource sharing access control strategies according to two-level STA models (TSTA) [1] are defined and these can be used to develop authority and authorization for every tenant.

3) The access control design for a geographic e-Science data and tools sharing platform based on the EasySaaS infrastructure [14] is described to demonstrate the TMS-ARBAC model.

This paper is organized as follows. Section 2 reviews the existing MTA infrastructures and the related access control models; Section 3 provides a formal definition of the TMS-ARBAC model, and then describes the resource sharing strategies for STA; Section 4 presents thorough security analysis of TMS-ARBAC; Section 5 applies TMS-ARBAC to the access control design for a geographic e-Science sharing application; and finally Section 6 concludes this paper.

2 Related work

This section reviews the MTA and STA in SaaS and their related security mechanisms.

2.1 MTA and STA in SaaS

Currently there are five ways to implement MTA [1]: integration with database; middleware approach; service-oriented SaaS; PaaS-based approach and object-oriented approach. Most of them focus on tenants' customizing and running their applications on a cloud platform.

However, Krebs et al. [16] pointed out that "multi-tenancy" lacked a clear definition. It sees the most existing MTA technologies, such as data center, virtualization and middleware sharing approaches, or PaaS-based MTA, are not real MTA, but multi-instance or multiple application deployment solutions are lacking in resources sharing and efficiency.

In MTA, most existing systems treat tenants as individual entities, fully isolated from other tenants. But SaaS promotes sharing. In real world, many cooperative, same-level, or same-type organizations may have similar application requirements or business processes. The relationship between two tenants needs to be discussed, but there are only a few works related published.

From the load balancing view, four kinds of "affinity" (non-affine, server-affine, cluster-affine and inter-cluster affine) are given to group different users of a tenant to process nodes for resource sharing and efficient-performance [16].

Maenhaut et al. [17] introduced a hierarchical model for the logical representation of the tenant tree and a mapping to the physical storage. But they focused on the efficiency and scalability of authentication data storage and access.

Keystone[2)] provided identity application programming interface (API) of hierarchical multi-tenancy for OpenStack, using domains and projects to build hierarchy of user groups and resources subsets .

Considering the hierarchical relationships among tenants in real life, a hierarchical multi-tenant pattern is introduced [6]. But none of them further discusses the customized resource sharing or isolation among tenants.

One successful MTA SaaS platform is Salesforce [18], running one application instance and one database schema to support multi-tenants (including enterprise-level tenants), using a metadata-driven software architecture, standard-based APIs and runtime application generator to enable multi-tenant customer relationship management (CRM) applications.

From the reusing of shared resources and easy customization view, STA [1] is proposed to model all kinds of hierarchical multi-tenancy with resources sharing among different-level tenants. It allows tenants to offer services for sub-tenant developers to customize their applications. Various STA models are defined with different customization models.

2.2 Access control in MTA

Security is an important topic in SaaS as all tenants share the same computing resources. MTA requires resource sharing, performance sharing, data privacy and application isolation at the same time. System security has been widely studied at different layers, e.g., network transition, system management and data storage. This paper focuses on authentication and authorization of MTA and STA.

To apply RBAC, to identify subjects, objects and permissions is necessary. In a SaaS system, a subject can be 1) a tenant; 2) a subtenant; 3) a user of either tenant applications or subtenant applications; an object can be any application or data components within the SaaS; and the permission is between subjects and objects within the SaaS system. The following approaches have been used to apply RBAC to SaaS

2) Hierarchical multitenancy in Keystone. http://raildo.me/hierarchical-multitenancy-in-openstack/, 2015

systems.

• Database schema and RBAC model

Traditional inforncation technology (IT) manufacturers develop data-centric PaaS-based SaaS, such as IBM[3] or Microsoft[4]. Multiple tenants are isolated by traditional database schema definition and RBAC model, sharing a data center.

Yaish et al. [4] proposed a multi-tenant access control model based on elastic extension tables (EET). "Parent-child user" and "group table" are used to describe user group and tenant group respectively. Roles are assigned to them for data isolation or sharing.

In these models, security strategies focus on data, but not on service sharing.

• RBAC-alike model for multi-tenant isolation

Li et al. [7] applied RBAC to SaaS systems and identified three problems: role name conflicts, cross-level management, and the isomerism of tenant's access control including the heterogeneous relations of roles and the heterogeneous constraints of permission assign. They proposed a S-RBAC model, in which access control is divided into two parts: tenant-level and system-level. Considering the role hierarchy and related constraints, they extended S-RBAC to H-RBAC model [8] with role delegation and time-constraints.

A tenant-based access control model T-Arbac [9] is proposed by adding "Tenant" into ARBAC model, and separating the functions of system administrators and tenant administrators. System resources are divided into sub-resource pools. Each tenant is assigned one certain pool, strictly isolating different tenants.

In Ref. [6], based on RBAC, "users" are extended to "unit", "single unit", "composite unit" and "user" to present the hierarchy of users.

Aiming at the shortcomings of ARBAC97 used in large organizations with many autonomous subsidiaries, N-RBAC [10] uses hierarchical namespace structure to arrange users and roles, more suitable for autonomous distributed role administration.

To meet the requirement of multi-hierarchies decentralized administration in large applications, a role-based hierarchical administrative model MHARBAC [5] is put forward, using role-tree to support top-down authorization. The inheritance relation in role hierarchies is removed, and new restrictions are added. Administrative scope is subdivided into user scope, role scope and permission scope.

• RBAC-alike model for multi-tenant sharing

Extensions of RBAC model have been proposed for collaborative authorization in clouds. Except those with centralized authority that are not suitable for the cloud, there are three ways to build collaborative authorization [11]: 1) delegation in RBAC on basis of individual user decisions; 2) federated identity and authorization services; 3) trust management into access control mechanisms.

For cross-tenant collaboration on outsourcing resources, a multi-tenant role-based access control (MT-RBAC) model family [12] is proposed to provide fine-grained authorization in a collaborative cloud environment by building trust relations among tenants. But it requires each tenant as "trustee" belongs to a single issuer as "truster", responsible for establishing the trust relation and adding trustee's users to truster's roles. But in SaaS, a shared resource may be composed of several components from different providers, which thus will not satisfy the assumption. And the truster's access control over trustee's users violates the autonomy of the trustee.

In SaaS ecosystem, service federation is established to allow a customer to use the services he subscribed in the federation across multiple organizations. Three major functions (single sing-on across services, account provisioning and secure delegation mechanism) are required for access control in the federation. IBM proposed an "open identity management framework" for SaaS ecosystem [13].

Salesforce[5] uses a unique organization identifier for each tenant. Server authentication and data encryption are used to make sure that data are safe to be accessed to by its owner. Three layered sharing designs are defined to expose different data sets to different sets of users. They are: object-level, field-level and record-level security. The NameDenorm table is used to represent parent-child relationships. Organization-wide, role-hierarchy and territory-hierarchy sharing can be used to define the sharing access to data records. Profiles are defined by a user's job function. Single sign-on-delegated authentication and federated authentication are provided for helping customers to login Salesforce within a corporation to use composed services.

Work has been done on service composition security solutions, and on the information flow problem in composite services from a service-oriented view [19]. But most of them focus on orchestration or execution of composite services. No

[3] Chong R. Designing a database for multi-tenancy on the cloud. http://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud/index.htm

[4] Chong F, Carraro G, Wolter R. Multi-tenant data architecture. http:// msdn.microsoft.com/en-us/library/aa479086.aspx

[5] Salesforce: security implementation guide. Version 31.0, 2014

work has been done on security problems in STA SaaS systems.

3 Tenant-based access control model for MTA and STA in service-oriented SaaS

In MTA or STA SaaS platforms, a tenant is an autonomous entity that can make many decisions including, keeping its own privacy and sharing its resources to its subtenants, allowing them to do further customization on what it provides, acting as a SaaS administrator in its own territory. This section introduces a tenant-based access control model called TMS-ARBAC based on ARBAC, for different-level tenants' isolation and sharing in STA service-oriented SaaS.

3.1 Overview of TMS-ARBAC

As shown in Fig. 2, TMS-ARBAC model adds three new entity components: tenants, chief administrative roles and federals on ARBAC. Tenant hierarchy is added to represent the hierarchy relationships between tenants and subtenants. But every component in TMS-ARBAC is not the same as defined in ARBAC.

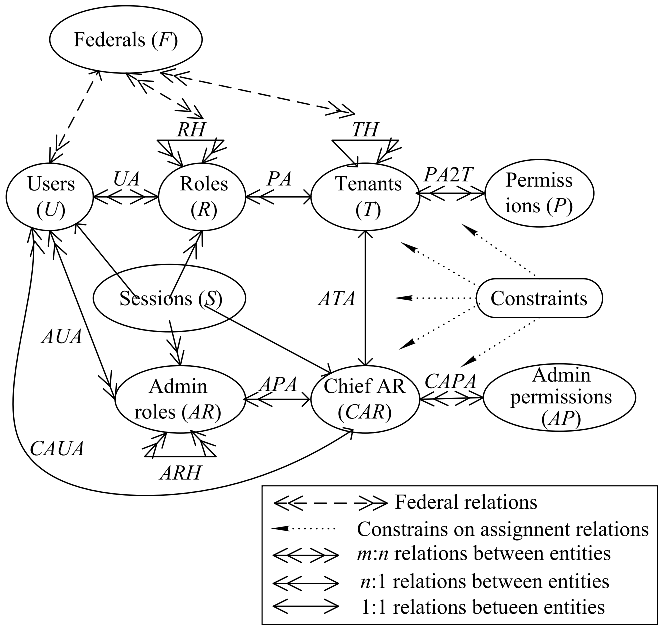


Fig. 2 The TMS-ARBAC Model

- Tenants (*T*) A tenant is an entity that establishes its own policies, and an application builder. It owns the resources rented from the system, and customizes its own application. It can create subtenants (*ST*) and users, authorizing what they can do with the resources the tenant owned or controlled. A unique tenant name is used to specify its autonomous area (including entities, resources, and permissions) from other tenants. The system administrator can neither access to a tenant's private data, nor do any changes within each tenant's permissions scope beyond service-level agreement (SLA). Every *ST* must have a parent-tenant, but a parent-tenant may have many *ST*s. A *ST* gets resources from its parent-tenant, and then it can do its own customization, and create its own users. In multi-level STA [1], a *ST* can create its own *ST*s.
- Roles (*R*) *R* has the same definition as in ARBAC. But unlike traditional RBAC, a role is narrowed down to a tenant scope, which is invisible to any other tenants normally. The role assignments come from its tenant's administrators, not from the SaaS administrator.
- Users (*U*) Both tenants and subtenants can create their users. The difference between "tenants" and "users" is that: "tenants" are organizations, but "users" are end users. Users are also narrowed down to a tenant's territory. Tenant or subtenant administrators authorize their end users by "roles". A user may have many roles. A role can be assigned to many users. A user's functions are distinguished by the roles it is assigned to, being an administrator, a tenant manager or a regular application user in its work zone.
- Federals (*F*) A federal is created for cross-tenant resource sharing, not including those between parent-child tenants. It is not a real organization or association, but a virtual entity. The assignments of tenants, roles and users in a federal will be further discussed in Sections 3.3 and 3.6.
- Permissions (*P*) *P* here has the same definition as in ARBAC. But in the service-oriented SaaS, resources and their access permissions need to be further subdivided, which will be further discussed in Section 3.5.
- Sessions (*S*) *S* here has the same definition as in RBAC. In a session, the roles assigned to the user can be activated. Note that in MT and ST SaaS, a user and the active roles of a session may be from the system, a tenant/subtenant or a federal. For tenants' privacy requirements, exclusive constraints should be used to avoid cross-tenant information purloin.
- Chief administrative roles (*CAR*) Administrative roles (*AR*) and administrative permissions (*AP*) are related to administrative work of tenants, roles, users and permission assignment, and autonomous area management, which will be further introduced in Section 3.4.

Table 1 lists all the abbreviations and their descriptions used in our model and strategies. All the relations labeled on the arrows in Fig. 2 will be formally defined in Section 3.3.

Table 1 Notations used in our model and strategies

Notation	Description
AA	Autonomous area
AP	Admin permission
APA	Permission to *AR* assign
ARH	Admin roles hierarchy
AR	Admin role
ATA	Tenant to *CAR* assign
AUA	*AR* to user assign
CAR	Chief admin role
CAP	Chief admin permission
CAPA	*CAR* to user assign
CSO	Chief security officer
CAUA	*CAR* to user assign
IP	Inherited permission
F	Federal
FUA	Outer role (*O_R*) to user assign
FR	Federal relationships
O_R	Outer role
P	Permission
PA	Permission to *RR* assign
PA2T	Permission to *AA* assign
PP	Private permissions
R	Roles
RH	Role hierarchy
RR	Regular role
S	Session
ST	Subtenant
T	Tenant
TH	Tenant hierarchy
TR	Tree relationship
U	User
UA	*RR* to user assign

3.2 Autonomous area tree management

System administrators, tenants and their subtenants, roles and users all live in one SaaS platform. Each entity has its own work domain. Unlike its identity in the traditional RBAC, the system administrator is no longer the most powerful authority. It creates tenants, giving tenants proper authorities to access to the SaaS platform resources according to SLA. It also revokes tenants' authorities and deletes them with SLA. Once a tenant is dropped, all its subtenants, roles, users and its privacies will be also dropped.

In MTA and STA, tenants are autonomous entities. Subtenants, roles and users should be defined in a tenant scope. For tenant isolation, roles cannot be granted to users of other tenants. Once a tenant is created, the authorization and authentication in the tenant scope cannot be interfered by the system administrator or other tenants.

To meet above requirement for tenants, we propose "autonomous area (AA)" to describe the security control area of one tenant. An AA is a named scope to restrict a tenant's RBAC security administration range. Each AA has a unique name. Every entity (such as a role, a user or a permission) of a tenant is unique in the tenant's AA with a unique entity name and its AA's name as a prefix. The access to control policies for one tenant can only be assigned to the entities in its AA.

Obviously, each AA has an independent name space. So, the entities in different AA can have the same name, which ensures a tenant to name its internal objects freely, without the consideration of name conflict with those in other tenants in MTA and STA environment.

As a tenant may have its subtenants, an AA may have its sub-AAs. All the AAs in MTA and STA environment form an AA-tree, representing the tenants' hierarchies, as Fig. 3 shows.

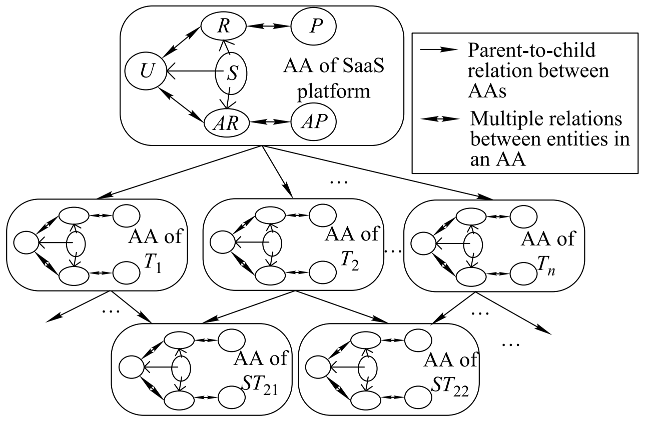


Fig. 3 An AA-tree structure of TMS-ARBAC model

However, from the point of view of tenants' hierarchies and AAs' creation order, the authority inheritance from the child-node to its parent-node is forbidden in the AA-tree. The objects in an AA cannot be visible from any other AA, even though the two AAs have a direct parent-child relationship between them, unless there is an explicit authorization of sharing, which will be discussed in Section 3.6.

3.3 TMS-ARBAC Model

The entity components and their relationships in one AA are shown in Fig. 4. And the TMS-ARBAC model is composed of a set of AAs.

The formal definitions of TMS-ARBAC model are as follows.

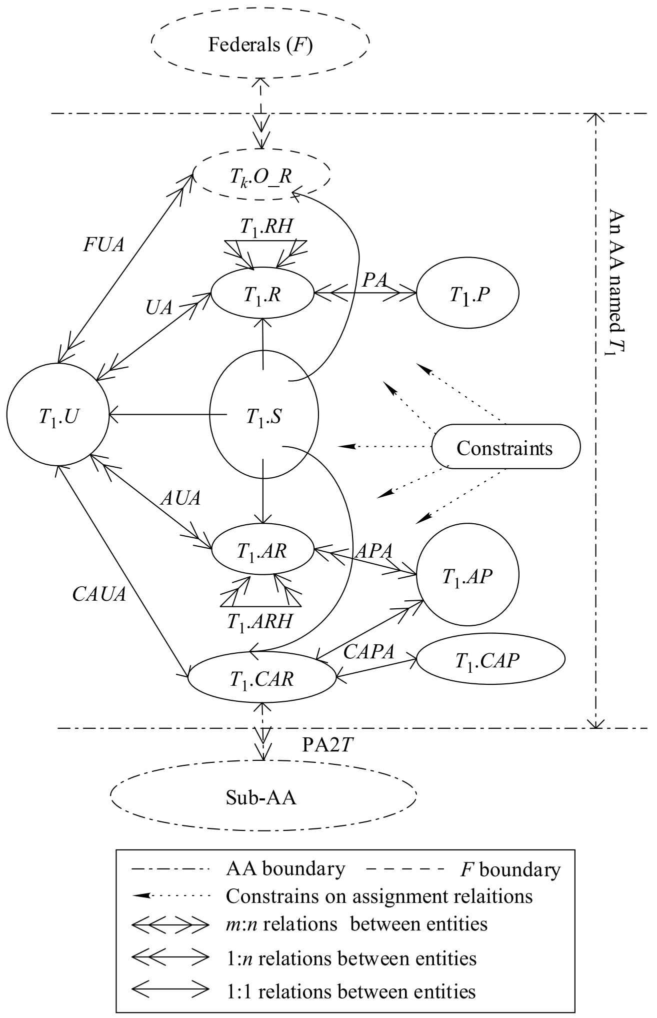


Fig. 4 Components and relationships in an AA-tree structure

Definition 1 An AA has the following components:

- U: a set of users created in current AA;
- R, AR and CAR: disjoint sets of (regular) roles, administrative roles and chief administrative role created in current AA;
- O_R: a set of outer roles shared by other AAs with the current AA;
- P, AP and CAP: disjoint sets of (regular) permissions, administrative permissions and chief administrative permissions in current AA;
- S: a set of sessions;
- $PA \subseteq P \times R$: many-to-many permission to regular role assignment relation;
- $PA2T \subseteq P \times A$: many-to-many permission to AA assignment relation;
- $APA \subseteq AP \times AR$: many-to-many permission to administrative role assignment relation;
- $CAPA \subseteq (AP \cup CAP) \times CAR$: many-to-one permission to chief administrative role assignment relation;
- $UA \subseteq U \times R$: many-to-many user to regular roles assignment relation;
- $FUA \subseteq U \times O\_R$: many-to-many user to O_R assignment relation. The inheritance or transitive relations among outer-roles are not allowed;
- $AUA \subseteq U \times AR$: many-to-many user to administrative role assignment relation;
- $CAUA \subseteq U \times CAR$: many-to-one user to chief administrative role assignment relation; in a tenant's AA, there is only one chief security officer (CSO);
- $RH \subseteq R \times R$: partially ordered role hierarchy;
- $ARH \subseteq AR \times AR$: partially ordered administrative role hierarchy (both hierarchies are written as $\geqslant$ in infix notation);
- user: $S \rightarrow U$: maps each session s_i to the single user user(s_i) (constant for the session's lifetime);
- roles: $S \rightarrow 2^{R \cup O\_R \cup AR \cup CAR}$: maps each session s_i to a set of roles, roles(s_i) $\subseteq \{r | (\exists r' \geqslant r)[(user(s_i), r') \in UA \cup AUA \cup FUA \cup CAUA]\}$ (which can change with time), and session s_i has the permissions $\bigcup_{r \in roles(s_i)} \{p | (r \in (R \cup AR) \wedge (\exists r'' \leqq r)[(p, r'') \in PA \cup APA]) \vee (r \in CAR \cup (p, r) \in CAPA) \vee (r \in O\_R \wedge (p, r) \in area(r).PA)\}$, where area(r) denotes the AA in which the role r is created.
- There is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden.
- The definitions of U, R, AR and S in an AA are just the same as ARBAC97 model, so in this paper we will not discuss them in detail.

Definition 2 The TMS-ARBAC model is denoted as a tuple $\langle AA, TR, FR \rangle$:

- $AA = \{a_1, a_2, \ldots, a_n\}$, where a_i is an AA, n is the cardinality of AA;
- If $n \leqslant 1$, TR is an empty set; else TR is a set of relationships denoted as $\langle a_i, a_j, P \rangle$, where if a_j is a sub-AA of a_i, then P is the permission set that a_j inherited from its parent-AA a_i. If not, then a_j is the parent-AA of a_i and P is the permission set that a_i grants to its parent a_j for resource sharing purpose. Each AA must have one and only one parent-AA, except the root AA.

- FR is a set of federal relationships. A federal f is defined as a five-tuple: $f = \langle F_{id}, C_a, F_m, F_o, F_c \rangle$, representing the resource sharing relationships between non-parent-child tenants. In which,

  F_{id} is the unique ID of the federal f.

  C_a is the chairman of f elected by all the federal members. It is also an AA.

  $F_m = \{a_1, a_2, \ldots, a_k\}, (k \geqslant 1)$, is a subset of AA, which represents the AAs joined in f.

  $F_o = \{\langle a_i, a_j, a_i.r \rangle\} \mid a_i \in F_m \wedge a_k \in F_m \wedge i \neq k \wedge a_i.r \in a_i.R\}$, which means AA a_i shares a role $a_i.r$ with another AA a_k in f.

  F_c is a group of constraints to f (such as time limit for f, exclusive constraints between two federals, etc.) and to the assignment constraints from f to its user.

According to Definition 2, there may be two types of relationship between two AAs: parent-child relationship and federal relationship.

The parent-child relationships always exist, so that every autonomous area in AAs constitutes a tree.

The federal relationship allows several AAs to unite a federal, and share roles between each other within the federal. The O_R of an AA a_i is a set of roles given by other AAs in the federal which a_i belongs to. F_{id} is added to O_R to distinguish outer roles assigned from the same AA of different federals.

$a_i.O\_R = \bigcup_{f \in FR} \{(a_j.r, f.F_{id}) \mid (\exists \langle a_i, a_j, a_j.r \rangle \in f.F_o \mid i \neq j)\}$.

3.4 Chief administrative role and chief administrative permissions

1) Chief administrative role (CAR)

In ARBAC97 model, administrative roles only have the permissions to control regular roles, and the administration of all the administrative roles and permissions is under control of a CSO, who has the highest authority in the system. But this premise no longer holds in the STA environment.

When RBAC mechanism is applied to an AA, the creation and assignment of the regular roles and administrative roles are supposed to be done by the CSO of the tenant, not the CSO of the SaaS platform. Furthermore, since tenants may have subtenants and subtenants are also autonomous, the parent-tenant should allow a subtenant to specify its own CSO to do administrative work in the subtenant' AA. In the multi-level STA environment [1], this process will continue recursively.

Thus, in the STA environment, each tenant has one CSO. There is not one global centralized CSO but many CSOs are distributed throughout the whole system. The administrative work includes creating users, RRs and ARs, permission assignment, and new sub-AAs or federals, which is continued dynamically in the life cycle of the SaaS system.

In TMS-RBAC model, we still use the RBAC mechanism to authorize CSOs. In each AA, except the original AR is defined in ARBAC97, we introduce a new administrative role, CAR, with highest authority in an AA. All the security administrative work of CSOs can be performed through CAR. Each AA has one CAR, in charge of developing and maintaining RBAC policies in the AA.

The difference between CAR and AR is that the former can be used to manage not only the regular roles but also the latter. This difference is achieved through a special set of permissions called CAP, as shown in Fig. 4.

When an AA is created, the CAR of this AA is created and granted the entire CAP automatically. Meanwhile, a default CSO user is created and the CAR is assigned to him automatically. In other words, the creation and assignment of CAR and CSO is involved in the automatic action of creating an AA. Then the CSO can be in charge of this AA.

No user could delete or modify the default user CSO or the role CAR because they are created by the system. But deleting an AA a_i will result in the deletion of a_i's CSO, CAR, and all the entities and resources in such AA. Taking into consideration that the parent-tenants should be in control of whether a subtenant could recruit subtenants, when creating an AA, the parent CAR member is allowed to specify whether the child CAR has the permissions to create or delete a sub-AA.

Constraint 1 There is one and ONLY one CAR in one AA.

2) Chief administrative permissions (CAP)

In TMS-ARBAC, the chief administrative permissions allow members of CAR c to perform some special administrative operations. These operations are described as follows.

- AddUser(c, u, a), which creates a user u in the AA a. The effect of the operation is: $a.U \leftarrow a.U \cup \{u\}$;
- DeleteUser(c, u, a), which deletes a user u from the AA a, where $u \in a.U$. The effect of the operation is: $a.U \leftarrow a.U - \{u\}$;
- AddAdminRole(c, ar, R_s, a), which creates an administrative role ar with immediate senior role set R_s in the AA a. The effect of the operation is: $a.AR \leftarrow a.AR \cup \{ar\}; a.ARH \leftarrow a.ARH \cup (\{ar\} \times R_s)$;

- DeleteAdminRole(c, ar, a), which deletes an administrative role ar from the AA a, where $ar \in a.AR$. All the related tuples in relations APA, AUA and ARH are also deleted. The effect of the operation is: $a.AR \leftarrow a.AR - \{ar\}$;
- CreateAutonomousArea(c, a_s, a, P_i, b), which creates a sub-AA a_s with immediate parent a. The permission set that a_s is inherited from a is P_i, where $P_i \subseteq a.P$. And b is a Boolean, denoting if a_s can create its sub-AA. The effect of the operation is:

  $AA \leftarrow AA \cup \{a_s\}$;

  $TR \leftarrow TR \cup \{\langle a, a_s, P_i\rangle\}$;

  $a_s.P \leftarrow P_i$;

  $a_s.U \leftarrow \{\text{CSO}\}$;

  $a_s.CAR \leftarrow \{CAR\}$;

  $a_s.CAUA \leftarrow \{(\text{CSO}, CAR)\}$;

  $a_s.CAPA \leftarrow a_s.CAP \times \{\text{CSO}\}$;

  if b is true, $a_s.CAP$ includes all operations in this section; otherwise, $a_s.CAP$ includes all operations in this section except CreateAutonomousArea, DeleteAutonomousArea, and ModifyAutonomousArea.

  The CAP is assigned to CAR by the system during the CreateAutonomousArea operation. They can only be assigned to CAR, and cannot be passed on to other ARs. In addition, since the CAP is assigned to CAR by the system, it is not allowed to be revoked from CAR by any user.
- DeleteAutonomousArea(c, a_s), which deletes a sub-AA a_s. The effect of the operation is:

  $AA \leftarrow AA - \downarrow a_s$;

  $TR \leftarrow TR - \{\langle a_i, a_{ij}, P\rangle \mid a_i \in \downarrow a_s\}$; $\downarrow a_s$ is the set of AAs that belongs to the subtree whose root is a_s.
- ModifyAutonomousArea(c, a, a_s, P_i, P_i'), which a modifies the sharing permission set of AA a_s from P_i' to P_i. The effect of the operation is:

  $TR \leftarrow (TR - \{\langle a, a_s, P_i'\rangle\}) \cup \{\langle a, a_s, P_i\rangle\}$;

  $a_s.P \leftarrow (a_s.P - P_i') \cup P_i$;

  $a_s.PA \leftarrow a_s.PA - \{(p, r) \mid p \in (P_i' - P_i)\}$.
- CreateFederal(c, a_i, cc, f, b), which creates a new federal f. And b is a Boolean, denoting if a_i is a member of f. The effect of the operation is:

  $f.C_a \leftarrow a_i$;

  if b=TRUE, $f.F_m \leftarrow \{a_i\}$; else $f.F_m \leftarrow \varnothing$;

  $f.F_o \leftarrow \varnothing$;

  $f.F_c \leftarrow \{cc\}$;

  $FR \leftarrow FR \cup \{f\}$.
- JoinFederal(c, a_i, f), which adds a new member a_i to federal f. The effect of the operation is:

  $f.F_m \leftarrow f.F_m \cup \{a_i\}$.
- ShareOuterRole$(c, a_i, a_j, R_o, cc, f)$, which shares a set of roles R_o in AA a_i with another AA a_j with a set of constraints cc, where $R_o \subseteq a_i.R$ and both a_i and a_j are members of federal f. The effect of the operation is:

  $f.F_o \leftarrow f.F_o \cup \{\langle a_i, a_j, a_i.r\rangle \mid a_i.r \in R_o\}$;

  $a_j.O\_R \leftarrow a_j.O\_R \cup \{(a_i.r, f) \mid a_i.r \in R_o\}$;

  $f.F_c \leftarrow f.F_c \cup \{cc\}$.

  Note: which role can be an O_R is decided by a_i.CSO; and the outer-roles assignment to users (FUA) of a_j is implemented by a_j.CSO.
- RevokeOuterRole(c, a_i, a_j, R_o, f), which revokes a set of roles R_o from another AA a_j in federal f, where $R_o \subseteq a_i.R$. The effect of the operation is:

  $a_j.O\_R \leftarrow a_j.O\_R - \{(a_i.r, f) \mid a_i.r \in R_o\}$;

  $f.F_c \leftarrow f.F_c - \{cc \mid cc.r \subseteq a_i.r \mid a_i.r \in R_o\}$;

  $f.F_o \leftarrow f.F_o - \{\langle a_i, a_j, a_i.r\rangle \mid a_i.r \in R_o\}$.

  Note: when an outer-role $a_i.r$ is deleted from a_j, the related FUA in a_j will be revoked by a_j.CSO.
- QuitFederal(c, a_i, f), which deletes a member a_i from federal f. The effect of the operation is:

  RevokeOuterRole(c, a_i, a_j, R_o, f), where $a_j \in f.F_m \wedge i \neq j \wedge R_o \subseteq a_i.R\}$;

  $f.F_o \leftarrow f.F_o - \{\langle a_x, a_y, a_x.r\rangle \mid a_x = a_i \vee a_y = a_i\}$;

  $f.F_m \leftarrow f.F_m - \{a_i\}$.
- DropFederal(c, a_i, f), which drops federal f created by a_i. Only the creator of federal f can do this operation. The effect of the operation is:

  QuitFederal(c, a_i, f), where $a_i = f.C_a$;

  $FR \leftarrow FR - \{f\}$.

Constraint 2 To avoid implicit privilege inheritance in role hierarchy, the partial order on O_Rs is NOT allowed both in a_i and a_j of $\langle a_i, a_j, a_i.r\rangle \in F_o (i \neq j)$.

Constraint 3 A set of outer roles $(a_i.r)$ can be assigned to an AA a_j as an O_R ONLY by their owner a_i. No grantee AA can assign an O_R to others.

Constraint 4 If an AA a joins two federals f_i and f_j, $a \in f_i.F_m \wedge a \in f_j.F_m$, any outer role $a.o\_r_i$ of f_i is invisible and invalid to f_j.

Constraint 5 To avoid the privacy leakage among different federals, exclusive constraints need to be added to federal user assignment, which means, NO user can simultaneously belongs to different federals in one session.

3.5 Permissions (P)

In an AA on a SaaS platform, permissions are not the same as in ARBAC97. Their differences are as follows.

1) The scope of permissions assigned to a role is restricted to its AA.

Permissions are related to resources. In the MTA and STA environment, each tenant only rents and owns part of the resources in the SaaS platform, so the permissions range for roles should also be bounded. In our model, the permissions to be granted from AR to RR are restricted to a subset of the AA's permission set.

2) The permissions consist of inherited and private permissions.

Because a tenant's customization and secret data are also resources and do not belong to the SaaS system, the permission set of each AA consists of two parts: permissions inherited from parent AA or shared from child AAs (called inherited permissions, IP) and permissions of private resources owned by the current AA (called private permissions, PP).

- For an AA a_i: $a_i.P = a_i.IP \cup a_i.PP$;
- $a_i.IP \subseteq parent(a_i).P \cup sub(a_i).P$, since $a_i.IP$ is inherited from a_i's parent-AA or shared by its sub-AAs;
- if a subset of $a_i.PP$ is granted to its sub-AA or parent-AA a_j, it turns to be an relative permission of a_j. For secret keeping, we require that it cannot be re-granted to other tenants from a_j.

Constraint 6 A set of private permissions ($a_i.PP$) can be assigned to an AA a_j ONLY by their owner a_i. a_j has no right to pass on $a_i.PP$ to other tenants.

A tenant may determine to revoke permission from its subtenant, and then all related permission-to-role assignments should be revoked in turn. Furthermore, if the subtenant has granted the permission to its subtenant, more complicated process is needed to ensure an exhaustive revocation.

3) Different resources have different permissions.

According to EasySaaS, system resources include GUIs, workflows, service components and Data (denoted as G, W, S, D). From the resources access control view, we prefer to divide these resources into two parts: AppComponent(G, W, S) and Data, for they have obvious difference on operations and data properties.

For STA applications and data access design, three questions should be considered.

- What permissions can tenants subtenants have?
- What permissions can tenants give to subtenants?
- What permissions can subtenants allow tenants to have?

Permissions that an AA can have and can grant to (or revoke from) others are concluded in Table 2.

Note that regular users of an AA are not application creators, and cannot do application customization. The application permissions to users are only "use" in application execution period, while the permissions to AA managers are "subscribe" and "customization" (the latter is default approved by our SaaS system) in application building period. "Upgraded" is a special permission from a component renter T_i to SaaS system, specifying whether T_i allows the auto-upgrade of the component.

When a tenant's private permissions are given to one of its subtenants, these permissions's privacy, re-grant of such IP is not permitted. From the data resources view, data should be detailed as "data" and "data space". Data sharing can be viewed as sharing a data space with shared data inside. But subtenants renting a data space from a tenant may cause a new access control problem.

When a new tenant T_i logs in the SaaS system, usually the system will give T_i an AA a_i with data space DS_i. When a subtenant T_{ij} of T_i is created, usually it will be given a sub-AA a_{ij} with data space DS_{ij} by a_i. $a_{ij}.DS_{ij}$ is partitioned

Table 2 Permissions that can be assigned to an AA

Permission property	AppComponents' permission set	DataSpace/Data's permisstion set
IP for public	Own/grant/revoke: $\{Us/S, TUs/TS, Upd\}$	Own/grant/revoke: $\{R, I, U, D, TR, TI, TU, TD\}$
$T_i.PP \rightarrow T_{ij}.IP$ for ST	Own: $\{Us/S, Upd\}$ with no re-grant	Own: $\{R, I, U, D\}$ with no re-grant
PP for privacy owner	Own: $\{Us, U, D, Upd\}$; Grant/Revoke: $\varnothing$ /$\{Us/S, Upd\}$	Own: $\{R, I, U, D, TR, TI, TU, TD\}$; Grant/revoke: $\varnothing/\{R, I, U, D\}$

Note: Us: use, S: subscribe, R: read, I: insert, U: update, D: delete, Upd: upgraded, T: transfer.

from its parent's data space, which satisfies $a_{ij}.DS_{ij} \subseteq a_i.DS_i$. Once DS_{ij} is given to a_{ij}, a_i cannot access to it without a_{ij}'s permission (expect in the data sharing case). But in the SaaS administrator's view, DS_{ij} still belongs to a_i. DS_{ij} becomes a "black area" of a_i, since a_i owns DS_{ij} but cannot control it. Adding data encryption on DS_{ij} by its last owner a_{ij} can help solve the "black area" problem. When a_{ij} is dropped, a_i re-gains the authority of DS_{ij}. Correspondingly, the encryption of a_{ij} on DS_{ij} is also removed. From the system's view, data space renting is a data space's multiple partition case.

From the AppComponent view, once a component is created and published on the SaaS platform, only its provider can update or upgrade the platform. Whether tenants who subscribe it do customization on it or not, there will be no change on the original component. But whether a tenant can or cannot do customization on it needs permission. For components reusability, application components are decomposed on SaaS platform as workflows and service components. A rented component can be a composite one. Once one of a composite component's sub-components is upgraded, this whole composite component should also be upgraded for its every renter. From the system view, application components renting is a component's multiple composition case.

To solve the access control problems in above two cases, two metadata tables should be added to record the renting history of decomposed data space or composite application components. They are:

- RCompositeInfoTable(*r_id*, *r_parent*, *r_owner*), records the composition or decomposition relations of resource components and data;
- RRenterTable(*r_id*, *granterAA*, *granteeAA*, *permissions*, *timeLimit*), records the resource permissions that the granter AA gives to the grantee AA with time limit.

3.6 Sharing strategies among tenants

As described above, in TMS-ARBAC model, two tenants (including a tenant and its subtenant) are strictly isolated from each other by their AAs. But in STA SaaS, sharing relationships between two tenants can be categorized as: tenant-subtenant relationship (T-ST), tenant-tenant relationship (T-T) and subtenant-subtenant relationship (ST-ST). Sharing strategy for each situation is given below.

1) T-ST sharing strategy

The T-ST sharing relationships are hierarchy sharing between a tenant and its subtenants. Five sub-models of two-level STA models (TSTA) [1] are defined to describe the sharing content between tenants and subtenants, which are: server-customers model (SC-TSTA), software-data model (SD-TSTA), master-slaves model (MS-TSTA), slave-masters model (SM-TSTA) and partner-partners STA (PP-TSTA). Using these sub-models, the resource sharing properties between a tenant and its subtenants are concluded in Table 3.

As shown in Table 3, for application components sharing, "customization" and "upgrade propagation" permissions are required in each STA model. Both of them should be set as default AppComponent sharing properties in MTA and STA SaaS. AppComponents' "sharing mode" is related to service component implementation for each renter. From the access control view, it is more related to how user authentication information passes among different sub-components or instances.

In TMS-ARBAC model, TR$\langle a_i, a_{ij}, P\rangle$ in the AA-tree is used to describe the parent-child resource hierarchy sharing relationships. The permission set P tells us what are shared between the parent-child tenants. The sharing between a tenant and its subtenants is realized by assigning shared resources and related permissions to the AA of the grantee. The sharing from parent to child AA is inheritance sharing, built with two operations: CreateAutonomousArea and ModifyAutonomousArea. The sharing from child to parent AA is the child's privacy sharing, implemented by ModifyAutonomousArea.

Automatic permission assignment operations between AA a_i and its sub-AA a_{ij} are described as follows.

- DSPAssignFromTtoST(a_i, a_{ij}, ds, P_i), which a_{ij} is given an isolated data space ds with permission P_i. The effect of the operation is:
  $a_{ij}.P \leftarrow a_{ij}.P \cup \{P_i\}$, where $P_i \subseteq \{(r\_id_{ij}, ds), R, I, U,$

Table 3 Sharing properties between parent-child tenants in TSTA models

Sharing properties	SC-TSTA	SD-TSTA	MS-TSTA	SM-TST	PP-TST
AppComp sharing	$T \rightarrow ST$	$T \rightarrow ST$	$T \rightarrow ST$	$T \rightarrow ST$	$T \leftrightarrow ST$
Sharing mode	one instance	one instance	multiple instances	multiple instances	multiple instances
Customization	ST	ST	ST	ST	T and ST
Data sharing	No	$T \rightarrow ST$	$T \rightarrow ST$	$T \rightarrow ST$	$T \leftrightarrow ST$
Upgrade propagated	Yes	Yes	Yes	Yes	Yes

$D, TR, TI, TU, TD\}$, where r_id_{ij} is the resource id of a_{ij} is not allowed to have its own sub-AA, then $P_i \subseteq \{(r\_id_{ij}, ds), R, I, U, D\}$;

insert into RCompositeInfoTable with tuple $(r\_id_{ij}, r\_id_i, r\_id_i.Owner)$, where r_id_i is the resource id of ds's parent data space;

insert into RRenterTable with tuple $(r\_id_{ij}, a_i, a_{ij}, P_i, timeLimit)$, a_{ij} becomes the current last renter of r_id_{ij}.

Note: though a_i's permissions on r_id_i still exist, ds is "locked" by a_{ij}'s encryption.

- DataSharePAssignFromTtoST$(a_i, a_{ij}, t_name, key_range[m, n], P_i)$, which allows a_{ij} to access a_i's table t_name within $key\_range[m, n]$ with a permission set P_i.

  $ds \leftarrow dataSpace(t\_name, key\_range[m, n])$;

  $a_{ij}.P \leftarrow a_{ij}.P \cup \{P_i\}$, where $P_i \subseteq \{ds, R, I, U, D, TR, TI, TU, TD\}$. If Data $(t\_name, range[m, n])$ is a_i's private data, then $P_i \subseteq \{ds, R, I, U, D\}$.

- DataSharePAssignFromSTtoT$(a_i, a_{ij}, t_name, key_range[m, n], P_i)$, which allows a_i to access a_{ij}'s table t_name within $key\_range[m, n]$ with permission P_i.

  $ds \leftarrow dataSpace(t\_name, key\_range[m, n])$;

  $a_i.P \leftarrow a_i.P \cup \{P_i\}$, where $P_i \subseteq \{ds, R, I, U, D\}$. Data$(t_name, range[m, n])$ is a_{ij}'s private data.

- AppCompSharePAssignFromTtoST(a_i, a_{ij}, acp, P_i), which allows a_{ij} to rent a_i's application component acp with a permission set P_i.

  $a_{ij}.P \leftarrow a_{ij}.P \cup \{P_i\}$, where $P_i \subseteq \{acp, Us, S, TUs, TS\}$. If acp is a_i's private application component, then $P_i \subseteq \{acp, Us, S\}$;

  insert into RRenterTable with tuple$(acp_id, a_i, a_{ij}, P_i, timeLimit)$, and acp_id is the resource id of acp. Every renter of each application component is recorded here, which helps components' upgrade and revocation.

- AppCompSharePAssignFromSTtoT(a_i, a_{ij}, acp, P_i), which allows a_{ij} to share its application component acp with a_i with a permission set P_i.

  $a_i.P \leftarrow a_i.P \cup \{P_i\}$, where $P_i \subseteq \{acp, Us, S\}$. The acp is a_{ij}'s private application component;

  insert into RRenterTable with tuple $(acp\_id, a_{ij}, a_i, P_i, timeLimit)$, and acp_id is the resource ID of acp.

2) T-T sharing strategy

In MTA and STA environment, tenants from different parents can unite a federal to share resources with each other, which means users of tenant T_i can span the isolation between tenants to access another tenant T_j's resources. We assume that a federal does not need to create special components for its all members. Or the federal will become a new tenant. So, we narrow down the federal sharing relationships to role sharing. In TMS-ARBAC model, T-T sharing is supported by federals and O_Rs, with CreateFederal, JoinFederal and ShareOuterRole operations. Whether a federal is a data-sharing or appComponent-sharing case is decided by the O_R definition.

Federal chairman, which is also an AA elected by other federal members, is responsible for the management of federal members and O_Rs, without interfering in any AA. The granter AA decides which role can be an O_R. And which user will be assigned the O_R is decided by the grantee AA with constraints F_c. Which O_R can be assigned to which specific AA or all federal members is defined with federal constraints F_c.

3) ST-ST sharing strategy

ST-ST refers to inside-tenant relationships. In our real life, there are branches of one company or organization having the same application requirements, but working on different regions.

If the sharing resources between two subtenants are their customized application components or their isolated data, which is a special case of T-T sharing, the ST-ST sharing strategy is federal sharing. The only difference between ST-ST federal and T-T federal is that in ST-ST federal their parent-AA is the chairman, but in T-T federal the chairman is also a member of the federal.

If not, the two subtenants must share resource directly from their parent. This can be easily solved by T-ST sharing strategy with inheritance sharing of the same data space or the same application components of their parent.

The difference between the federal sharing (FR in AA-tree) and parent-child sharing (TR in AA-tree) is that: the latter takes the granularity of permission, so that the administrators of the receiver AAs can combine the permissions into different roles flexibly to meet the access control requirement of the organizations; so that local users can only access to outer resoarnes through specifecd roles with strict constraints to achieve more reliable security.

4 Evaluation of TMS-ARBAC

4.1 Security analysis of TMS-ARBAC

Considering an AA as a distributed autonomous organization in the cloud, the sharing security among AAs turns to be a secure interoperation problem [20–22] in a STA SaaS environment.

From the access control view, the security attributes of a system are expressed with an access control list (ACL) [23]. The security aspect of interoperation is represented by access rights across AAs. There are two kinds of interoperation among AAs, TR and FR, in TMS-ARBAC. The AA secure interoperation problem can be defined as: given the access control lists for each AA in a MTA and STA SaaS system, an interoperation TR and/or FR, which is a set of access control entries where, for each entry, the subject and the object belong to different AAs. The general problem is to decide if there exists any access that is not permitted in one AA but is permitted as a result of interoperation, investing and removing security violations while maintaining a reasonable level of interoperation.

According to TMS-ARBAC model, each AA has an ACL = $\{\cup IP, \cup PP, \cup O\_R\}$, denoting a set of IP given by its parent-AA or sub-AAs, its PP and O_Rs given by the federals it joins. We define $(a_i, a_j, p/o\_r)$ as an access from AA a_i to a_j's resources through some sharing mode (permissions or O_Rs). Two kinds of security interoperation problems should be addressed: hierarchy sharing interoperation and federal sharing interoperation.

With FR and TR, class interoperations in an AA-tree are shown in Fig. 5.

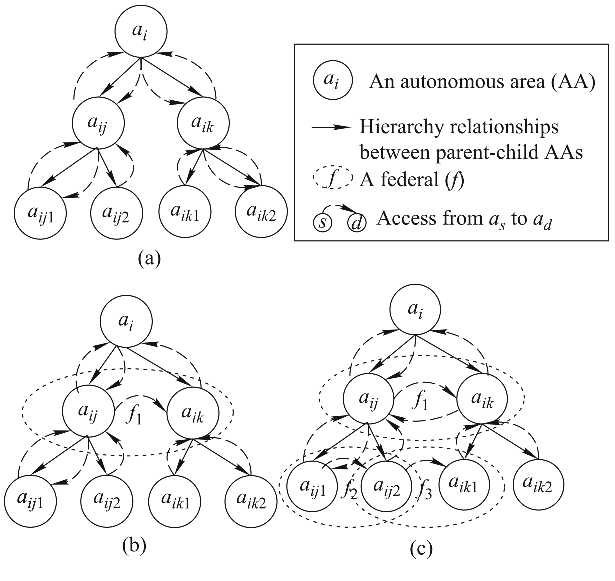

a_i
a_{ij}
a_{ik}
a_{ij1}
a_{ij2}
a_{ik1}
a_{ik2}
(a)
a_i An autonomous area (AA)
Hierarchy relationships between parent-child AAs
f A federal (f)
s d Access from a_s to a_d
f_1
(b)
f_2
f_3
(c)

Fig. 5 Interoperations in the AA-tree. (a) Hierarchy sharing access in an AA-tree; (b) hierarchy sharing and federal sharing access; (c) two federals sharing access with one same AA

- Security analysis of hierarchy sharing interoperation among AAs

Assume an AA a_i has $ACL_i = \{\cup IP_i, \cup PP_i, \cup O\_R_i\}$, while $\cup O\_R_i = \emptyset$.

Case 1 If $\cup IP_i \neq \emptyset, PP_i = \emptyset$, all resources that can be accessed by a_i are inherited from its parent-AA or shared by its sub-AA. Class $\cup IP_i$ by resource types are as follows.

Case 1-1: data spaces exclusive access permissions from parent to child AA (that is the data space allocating case). The owner of such data space (denoted as ds_i) is changed from its parent-AA to a_i. So, it becomes an exclusive resource of a_i. If a_i has a subtenant a_{ij}, and allocates some free data spaces (denoted as ds_{ij}) of a_i to a_{ij}, then $a_i.DS = ds_i - ds_{ij}$, which is still an exclusive resource of a_i.

Case 1-2: data sharing permissions between parent-child AAs. That is, a_i's access to the data owned by its parent-AA/sub-AA is a permitted access. If such data are secret data of the granter-AA, the grantee-AA cannot re-grant them to other AAs.

Case 1-3: AppComponent sharing permissions between parent-child AAs. If the AppComponent is a public component without secret or personal information, every AA can use it with permissions. If not, only an individual instance of it will be given to a_i. No one but only the component owner can modify the original components. No AA can get others' component privacy. If such component is a customized component of the granter-AA, the grantee-AA has no right to re-grant it to others.

So, a_i's access to the resources shared, no matter they are private or not for its parent-AA or sub-AA, is either exclusive or authorized. For a_i's parent-AA, the resources it gives to its subtenant are either shared with its permissions or exclusively allocated to a_i. No security violations exist in both a_i and its parent-AA.

Case 2 If $\cup IP_i \neq \emptyset, \cup PP_i \neq \emptyset$, a_i has its own private data or customized application components. Here leave internal AA administration aside. a_i either monopolizes the private resources or shares some of them with its parent-AA or sub-AA (as described in Case 1-2 or 1-3) with permissions. Constraint 6 is used to prevent privacy leaking outside a_i and its authorized relative-tenants.

Case 3 Considering an AA-tree with height =3, as shown in Fig. 5(a), the interoperation between parent-child AAs is all implemented by granting permissions of one AA to the other. Unlike roles, permissions have no hierarchies or partial ordering relation. No extra access path can be created for cross parent-child AAs' access.

Meanwhile, shared resources (except secret or private resources) can be conditionally transmitted throughout the whole AA-tree. For example, as shown in Fig. 5(a), a_{ij1} can share a_{ij} with its data access permission p, denoted as (a_{ij}, a_{ij1}, p). Supposed such data is not secret but worthy, $(a_i, a_{ij}, p')(p' \subseteq p)$, and a_{ik} can inherit part of such data from $a_i(a_{ik}, a_i, p'')(p'' \subseteq p')$, etc. This demonstrates that resources can be flexible shared in an AA-tree (a hierarchical organization), satisfying the application requirement of the STA SaaS.

However, we find a kind of security violations in such hierarchy sharing interoperation. Using above example, if what are shared are the same data stored in the same data space (which means no data copy), then a_i and a_{ik} can access the data of a_{ij1} without its authorization. But such data shared are not secret, or the authority transmission will be forbidden with Constraint 6. Likewise, an AppComponent can also be cross-level shared without direct permissions. But such component can only be public, without private information leaking. We can also use mandatory access control or trust domain management to avoid such kind of security violation.

• Security analysis of federal sharing interoperation among AAs

Assume AA a_i has $ACL_i = \{\cup IP_i, \cup PP_i, \cup O\_R_i\}$, while $\cup O\_R_i \neq \varnothing$, which means a_i may join several federals to share resources of other AAs.

Case 4 Assume there is only one federal in an AA-tree, as shown in Fig. 5(b), a federal f_1 has two AAs, a_{ij} and a_{ik}, with an access (a_{ij}, a_{ik}, O_R). If (a_{ij}, a_i, p_1) and (a_i, a_{ik}, p_2) $(p_1 \subseteq p_2)$ exist (the same case as the above security violation), assume that O_R and p_1 are both assigned to one user u in a_{ij}, then u has two ways to access a_{ik}. Can both O_R and p_1 be assigned on one resource of a_{ik}?

No. Because in this case, a_{ij} and a_{ik} are siblings, what sharing between them with federal sharing mode must be secret, as defined in Section 3.6 3) ST-ST sharing strategy. But the other access control list is an authority transmission way. The sharing resources must not be secret or private. So, there is no security interoperation violation in this case.

Case 5 If we add a new access (a_{ik}, a_{ij}, O_R') in f_1, may there be a security violation in f_1?

No. According to Constraint 2, the partial order on O_Rs is cut off both in a_{ij} and a_{ik}. No access loop can be created between a_{ik} and a_{ij}. And the user to whom the O_R is assigned is decided only by the grantee AA with O_R assigning constraints in f_1, to avoid the O_R to be assigned to any untrustworthy user.

Case 6 Assume there are two federals with one same AA in an AA-tree, as shown in Fig. 5(c), federal f_2 with federal members a_{ij1} and a_{ij2}, f_3 with a_{ij2} and a_{ik1}. If accesses $(a_{ij1}, a_{ij2}, O\_R_1)$ and $(a_{ij2}, a_{ik1}, O\_R_2)$ individually exist in f_2 and f_3, may there be an access from a_{ij1} to a_{ik1} derived?

Yes. Though with Constraints 3 and 4, a_{ij2} cannot assign O_R_2 as an outer role to a_{ij1}, but if a_{ij2} can read a_{ik1}'s data and write them down in its own data space, then it can transfer them to a_{ij1} with O_R_1. We use Constraint 5 to avoid direct transmission from a_{ik1} to a_{ij2} then to a_{ij1}. But this illegal access may still implicitly exist. We can use separation of duties, such as Chinese Wall security policy, and trust management to deal with such violation problem.

4.2 The characteristics of TMS-ARBAC

The typical characteristic of TMS-ARBAC is to reconcile the demand of tenant isolation and resource sharing in STA SaaS.

1) The isolation and autonomy of every tenant

- Each tenant has its isolated work area AA. Resources, roles and users are all restricted in such work area. So, an AA builds a strict isolation wall from other tenants.
- System-level and tenant-level administrations are clearly separated. The system administrators care more about system resources allocation and management, but tenant administrators care about how to use their own resources. The SaaS system is also an AA. Each AA has its CAR and CSO. Different job functions can be clearly defined and executed in different AAs.
- Each tenant establishes its access control policies with application requirements in its AA. System administrators cannot interfere with a tenant's internal management. A tenant cannot interfere with its subtenants' internal affairs. Only the resource allocation and revocation permissions obey the AA hierarchies.

2) Resource sharing for tenants

- Federal sharing Subscribed resources are shared between unrelated tenants or sibling tenants in one federal. The ownerships of such resources are not changed. Security is ensured by federal trust and O_R constraints.
- Inheritance sharing System resources inherit from parent to child tenants. The ownership of such resource may be changed from the parent-tenant to the subtenant, such as the data space re-allocation case. Once the ownership is changed, the former owner loses its authority to the resource till the revocation of the subtenant. If not changed, the grantee has no right to update the original resources.
- Privacy sharing between tenants Private resources,

including application customization and secret data, are privately shared between parent-child tenants or sibling tenants, which is a kind of private resource reuse. Privacy ownership cannot be changed. Privacy access privileges can only be issued by its owner. The grantees are forbidden to spread others' privacy, which keeps secret from spreading.

The model function comparison between TMS-ARBAC and four typical MTA access control models (H-RBAC [8], T-arbac [9], access control on EET [4] and Salesforce[6]) are shown in Table 4.

Table 4 STA and MTA access control models comparison

Model	A	B	C	D	E	F
Function separation between system administrators and Tenants	✓	✓	✓	✓	✓	✓
Tenant self-management	✓	✓	✓	✓	✓	✓
Tenant creation by its parent tenant	✓	×	×	✓	✓	✓
Subtenant self-management	✓	×	×	×	✓	/
Roles restricted in tenant scope	✓	✓	✓	✓	✓	✓
Tenant resource isolation	✓	✓	✓	✓	✓	✓
Resource federal sharing between tenants	✓	/	/	✓	/	✓
Resource inherited between parent-child tenants	✓	×	×	✓	/	✓
Subtenants do re-customization	✓	/	/	×	/	✓
Privacy sharing between tenants	✓	×	×	×	×	/

Note: A = TMS-ARBAC; B = H-RBAC; C = T-arbac; D = EET access control; E = N-RBAC; F = Salesforce security; ✓ = support; × = not support; / = not mentioned

From the comparison above, TMS-ARBAC and the Salesforce security solution are the best to satisfy the STA access control requirement. But Salesforce only focus on the CRM field; and it handles MTA and STA in its special way on Force.com. TMS-ARBAC treats each subtenant as autonomous individual, and handles various sharing relationships with loose-coupled tree structure, which is a generic, flexible and decentralized model that can be easily applied to cloud environment.

This model can also be used on other system environments besides cloud, only if the environment has multiple users, numerous resources (e.g., service, application components, and/or data), and has the access control requirement to let each user or user group exclusively own some resources, customize its own application and/or flexibly share some single resource or resource package with others.

5 A case study

This paper applies the TMS-ARBAC model to the access control of a geographic e-Science data and tools sharing cloud platform based on EasySaaS.

1) The geographic e-Science society

This society holds e-Science data from various geographic areas, including geo-data, weather data, hydrologic data and geological disaster data. The organization of such geographic e-Science society is shown in Fig. 6.

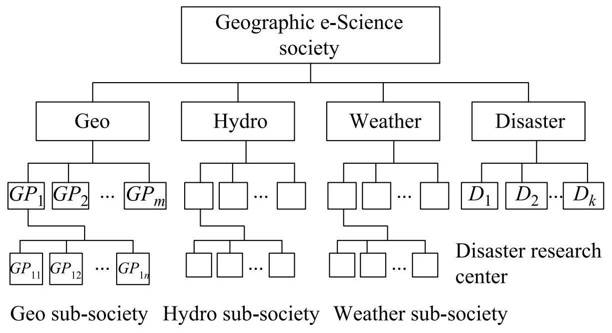


Fig. 6 The geographic e-Science society organization

Each sub-society is an independent organization or enterprise, which may have hierarchical branches, divided as one national headquarter, several province-level departments and city-level departments. Different from above sub-societies, the geo-disaster research center is divided by the job functions of departments, such as: data collection department, data analysis department and disaster prediction department.

Each sub-society has its corresponding data services, including regular data services (such as data upload, download and cleansing, etc.) and special services (such as vector data slicing algorithms and various data mining algorithms).

2) Resource isolation and sharing requirements

- Isolation requirements
  I-1: different research fields or sub-societies are independent enterprises or organizations; the geo-disaster research center is an independent organization too.
  I-2: in a sub-society, such as in the geo-society, each province-level branch is an autonomous entity, which owns several city-level sub-branches with similar job functions distributed in different area.
  I-3: every sub-society, branch or sub-branch has its own data space.
- Sharing requirements
  S-1: common components global sharing. Such as data upload, and download services are shared in the whole society.
  S-2: downward application components sharing. In one

[6] Designing record access for enterprise scale. https://twitter.com/ salesforcedocs, 2015

sub-society, the national headquarter may share its application components with its province branches, and a province branch may share its components with its city branches. Service customization in national-level and provincial-level is permitted.

S-3: sibling branches. Two sibling branches (province-level or city- level) may have similar business routines, sharing the similar service components with their own customization.

S-4: upward data sharing. Each branch submits its data to its upper department, e.g., a city-level department submits its data to the province- level department it belongs to, and a province-level department submits its data to its national-level department.

S-5: federal data sharing. The geo-disaster center can gather science data from different research fields for geo-disaster analysis and prediction. And its research results can be published to the related fields.

From the view of S-2 and S-4 sharing requirements, this geographic e-Science SaaS system is a slave-master STA case.

3) Access control strategies for the geographic e-Science SaaS cloud

According to above application requirements, Fig. 7 shows the geographic e-Science SaaS cloud infrastructure. The access control strategies are given out as follows.

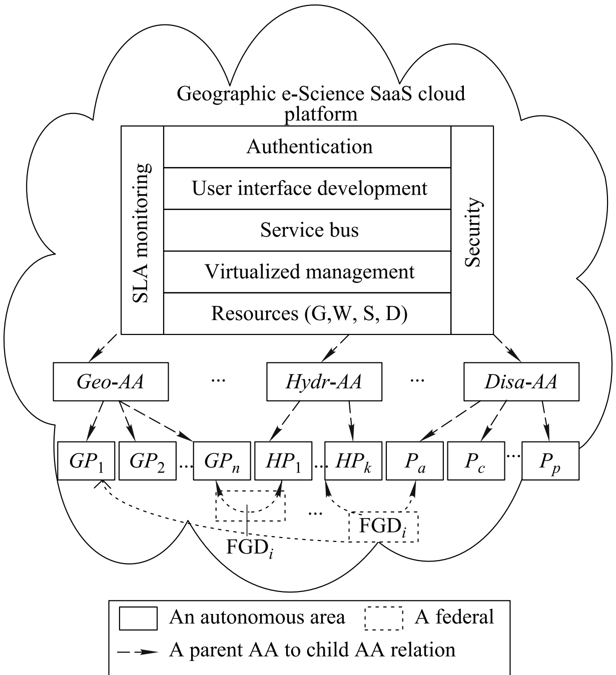


Fig. 7 The geographic e-Science SaaS cloud infrastructure

- Isolation strategy: using AAs to isolate organizations/enterprises; in one AA, using roles assignments to limit user's working zone.

  For I-1, each sub-society is built as a tenant in the SaaS platform. Corresponding AA is given to each tenant: *Geo-AA*, *Hydr-AA*, *Weath-AA*, etc. The geo-disaster center is also a tenant: *Disa-AA*. Roles are assigned to the national headquarter managers of an AA to split their working zones.

  For I-2, each province-level branch in one sub-society is built as a subtenant. For example, different province-level departments are subtenants of Geo-AA. Their AAs are denoted as: GP_1, GP_2, etc.

  For I-2 and I-3, each sub-branch of a province-level branch can also be defined as a subtenant. But since these city-branches may have the same application logic, only their data space needs to be separated. This is a traditional MTA case. We use roles and users assignments to share service components and separate data for such sub-sub-branches.

  Disa-AA has different branches with different job functions, but every branch may use the same data. Since few application components sharing exist between branches, such branches are not defined as subtenants, but using roles and users assignments to control their access to service components.

- Sharing strategies

  For S-1, a public resource mark is given for global sharing. For every tenant to access the common components freely, a special mark "PUBLIC" is given to the resource's access_Property.

  For S-2, the T-to-ST application component sharing strategy is applied. Proper application components are shared from an AA to its sub-AAs: AppCompSharePAssignFromTtoST (a_i, a_{ij}, acp, P_i);

  For S-3, the ST-to-ST federal sharing strategy is used for sharing between sibling tenants. Customized application components are shared between sibling subtenants: ShareOuterRole $(a_i.CSO, a_{ij}, a_{ik}, a_{ij}.r, cc, f)$, where a_{ij}.r is a set of roles to access the customized components. City-level branch components sharing can be realized by roles assignments.

  For S-4, the ST-to-T data sharing strategy is applied. Data components are shared from sub-AAs to their parent AA: DataSharePAssignFromSTtoT$(a_i, a_{ij}, t_name, key_range[m,n], P_i)$.

  For S-5, the T-to-T federal sharing strategy is used for sharing between unrelated tenants. Federals are created between Disa-AA and other sub-society AAs. O_Rs are given from sub-society AAs to Disa-AA for data sharing and geographic disaster analysis and prediction: ShareOuterRole$(c, a_i, a_j, a_i.r, cc, f)$, where $a_i.r$ is a set of O_Rs for Disa-AA to access their data; and Disa-AA also creates O_Rs for other sub-society AAs to access to its research results.

4) Access control sub-system design in the geographic e-Science SaaS cloud platform

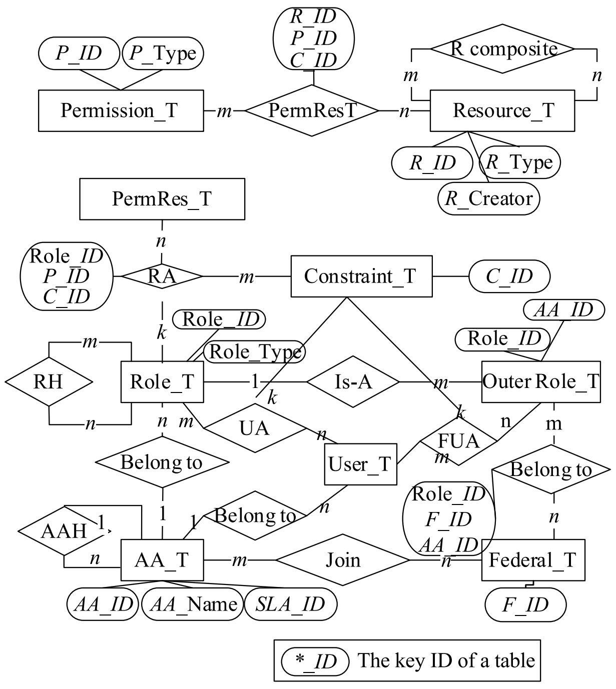


Fig. 8 E-R diagram design for access control in STA SaaS

• Access control database design

The metadata database schema for general access control in STA SaaS is given in Fig. 8.

• Access control module design

There are two kinds of authorization: system authorization and user authorization in MTA and STA SaaS platforms.

System authorization creates AA and the entities in an AA and defines authorization scope for each authorized entities, such as users, roles (RR and AR), permissions (AP and P), and resources. User authorization is the user assignments granted by the ARs in an AA. In an AA, a tenant manager is different from a user. The former can subscribe components from other AAs and do customization, but the later can only use the components. So, their permissions to application components are different.

As shown in Fig. 9, when a user enters the system, its username and password are sent to the authentication center. A tag with the user type, authority scope and role sets information is returned for the user's related resource configuration in the system.

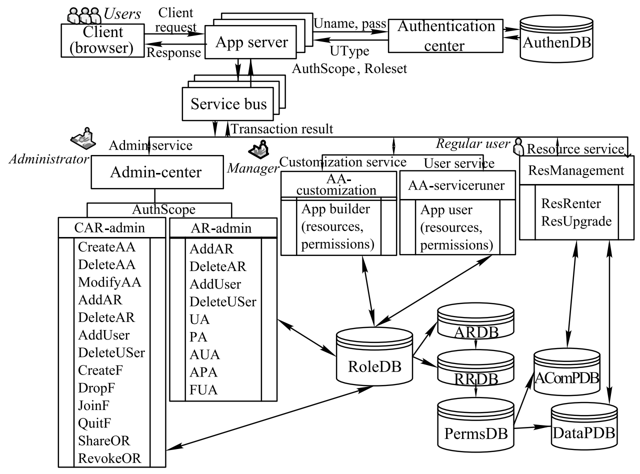


Fig. 9 The access control workflows in the geographic e-Science SaaS

There are three kinds of users in the system: security administrators, application managers and application users. Every user has his own work zone. Role sets tell what the user can access in his work zone.

Security administrations and application customization are added into the SaaS platform as administration and customization services, deployed on the service bus with regular application services and resource services.

According to the user type, a user will be delivered to corresponding services. Authority scope decides the place where AA the user works. Different resources and permissions are configured for the user according to their role sets.

The access control module of the geographic e-Science SaaS platform is given out in Fig. 10.

A security management center is built to manage the access control of the whole platform, including several distributed authentication servers, access control servers, dynamic constraint control servers and an access control database center.

Service bus scheduler provides corresponding services and other resources with the roles and permissions of the user. For system scalability, tasks can be divided by different roles (such as *RR*, *AR* and *CAR*), different permissions (such as *P*, *AP* and *CAP*) and different AA scopes. Parallel execution is provided to ensure system performance and efficiency.

6 Conclusion and future work

This paper analyzes the access control requirements of MTA

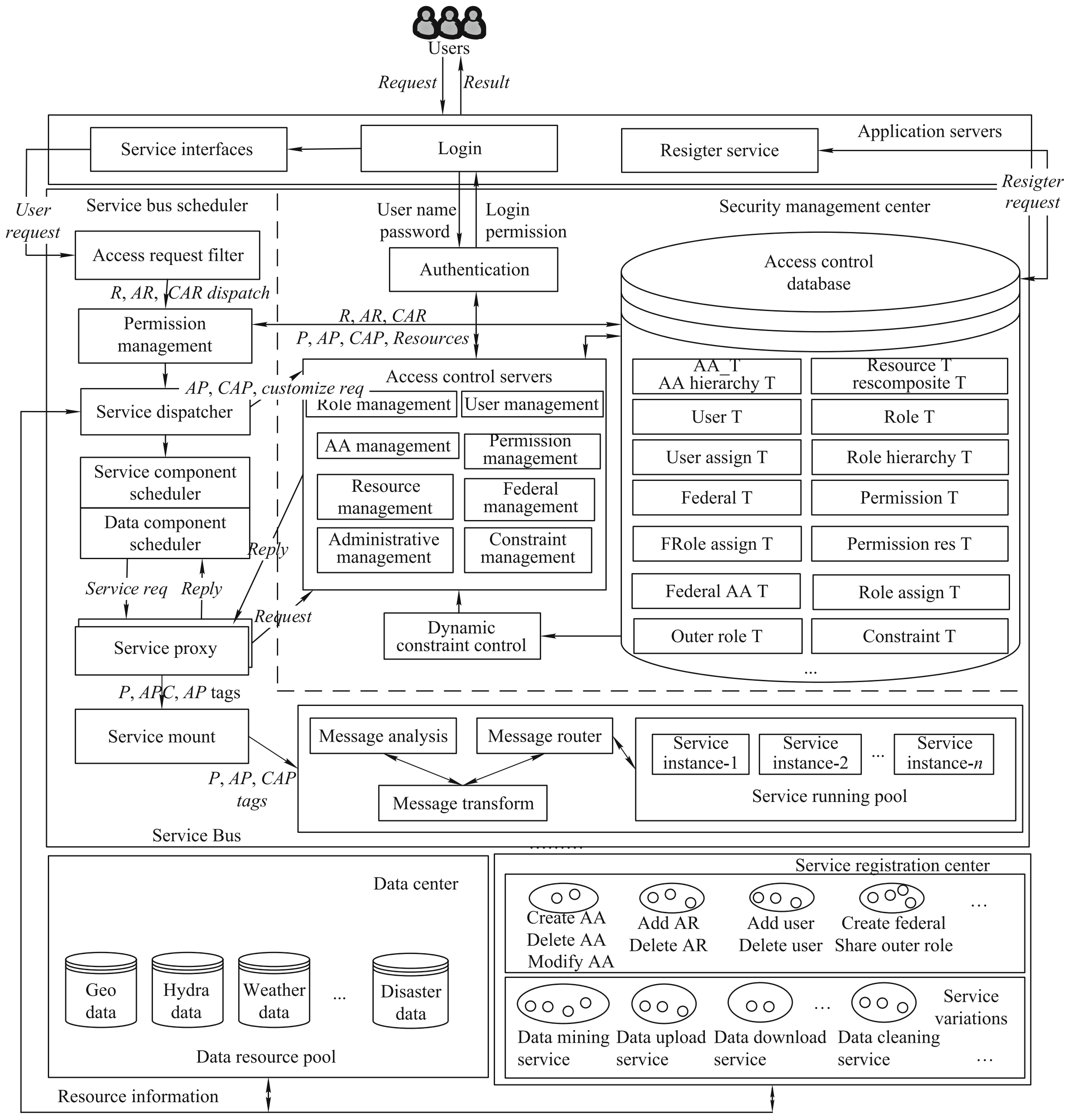


Fig. 10 The access control module design (Req: request)

and STA SaaS platforms, emphasizing the isolation and sharing issues among tenants, and comes up with a formal definition of a new TMS-ARBAC model. AAs are used to isolate each tenant or subtenant, and restrict roles and users to their work zones in an AA. Each tenant is self-managed and can do customization easily with the variation of application requirements. An AA-tree is a loosely-coupled and highly extensible structure, describing the tenants' hierarchy and various sharing relationships. For the access control on service-oriented SaaS, "black area" problem in sub-AA configuration, resource and permission category definitions and different sharing properties in different STA modes are also considered. General authorization operations on an AA and different resource sharing strategies are given and implemented on a geographic e-Science SaaS cloud platform to demonstrate the model.

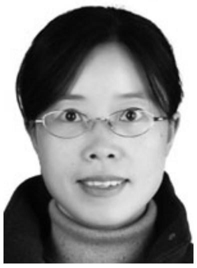
Currently, we are working on various access control strategies for different STA models. Future research work will focus on the scalability and distribution characteristics of MTA and STA SaaS system to improve efficiency of authorization and authentication.

Acknowledgements This paper was based on projects sponsored by the State Scholarship of the China Scholarship Council.

References

1. Tsai W T, Zhong P. Multi-tenancy and sub-tenancy architecture in Software-as-a-Service (SaaS). In: Proceedings of the 8th IEEE International Symposium on Service Oriented System Engineering. 2014, 128–139
2. Sandhu R S, Coyne E J, Feinstein H, Youman C. Role-based access control models. IEEE Computer, 1996, 29(2): 38–47
3. Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security, 1999, 2(1): 105–135
4. Yaish H, Goyal M. Multi-tenant database access control. In: Proceedings of International Conference on Computational Science and Engineering. 2013, 870–877
5. Zhong H, Wang W, Yan G, Lei Y. A role-based hierarchical administrative model. In: Proceedings of International Conference on Computational Intelligence and Software Engineering. 2009, 1–4
6. Bien N H, Thu T D. Hierarchical multi-tenant pattern. In: Proceedings of International Conference on Computing, Management and Telecommunications. 2014, 157–164
7. Li D, Liu C, Wei Q, Liu Z, Liu B. RBAC-based access control for SaaS systems. In: Proceedings of the 2nd International Conference on Information Engineering and Computer Science. 2010, 1–4
8. Li D, Liu C, Liu B. H-RBAC: a hierarchical access control model for SaaS systems. International Journal of Modern Education and Computer Science, 2011, 3(5): 47–53
9. Cao J, Li P, Zhu Q, Qian P. A tenant-based access control model T-Arbac. Computer Science and Application, 2013, 3: 173–179
10. Xia L, Jing J. An administrative model for role-based access control using hierarchical namespace. Journal of Computer Research and Development, 2007, 44(12): 2020–2027
11. Tang B, Sandhu R, Li Q. Multi-tenancy authorization models for collaborative cloud services. In: Proceedings of International Conference on Collaboration Technologies and Systems. 2013, 132–138
12. Tang B, Li Q, Sandhu R. A multi-tenant RBAC model for collaborative cloud services. In: Proceedings of the 11th Annual International Conference on Privacy, Security and Trust. 2013, 229–238
13. Wang B, Huang H, Liu X, Xu J. Open identity management framework for SaaS ecosystem. In: Proceedings of IEEE International Conference on e-Business Engineering. 2009, 512–517
14. Tsai W T, Huang Y, Shao Q H. EasySaaS: a SaaS development framework. In: Proceedings of IEEE International Conference on Service-Oriented Computing and Applications. 2011, 1–4
15. Masood R, Shibli M A, Ghazi Y, Kanwal A, Ali A. Cloud authorization: exploring techniques and approach towards effective access control framework. Frontiers of Computer Science, 2015, 9(2): 297–321
16. Krebs R, Momm C, Kounev S. Architectural concerns in multi-tenant SaaS applications. In: Proceedings of the 2nd International Conference on Cloud Computing and Service Science. 2012, 426–431
17. Maenhaut P J, Moens H, Decat M, Bogaerts J, Lagaisse B, Joosen W, Ongenae V, De Truck F. Characterizing the performance of tenant data management in multi-tenant cloud authorization systems. In: Proceedings of IEEE/IFIP Network Operations and Management Symposium. 2014, 1–8
18. Weissman C D, Bobrowski S. The design of the Force.com multitenant Internet application development platform. In: Proceedings of ACM SIGMOD International Conference on Management of Data. 2009, 889–896
19. Wei S, Yen I L, Thuraisingham B, Bertinod E. Security-aware service composition with fine-grained information flow control. IEEE Transactions on Service Computing, 2013, 6(3): 330–343
20. Gong L, Qian X L. The complexity and composability of security interoperation. In: Proceedings of IEEE Symposium on Research in Security and Privacy. 1994, 190–200
21. Gong L, Qian X L. Cumputational issues in secure interoperation. IEEE Transactions on Software Engineering, 1996, 22(1): 43–52
22. Shafiq B, Joshi J B D, Bertino E, Ghafoor A. Secure interoperation in a multi-domain environment employing RBAC policies. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(11): 1557–1577
23. Lampson B W. Protection. ACM Operating Systems Review, 1974, 8(1): 18–24

Qiong Zuo is a lecturer of the School of Computer Science and Technology, Huazhong University of Science and Technology, China and was a visiting scholar in Arizona State University, USA from 2014 to 2015. Her research interests are database system management, cloud computing and big data management.



Meiyi Xie is a lecturer of the School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her research interests are in information security, including database security, intrusion tolerance, cryptography and privacy-preserving data publishing.

Guanqiu Qi received his PhD in Schools of Computing, Informatics, and Decision Systems Engineering, Arizona State University, USA in 2015. His research interests are service-oriented architecture, Software-as-a-Service, Testing-as-a-Service and big data testing.

Hong Zhu is a professor of School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her research interests are in data security, including database security, cryptography and privacy-preserving data publishing.