applied sciences

MDPI

Article

Role-Based Access Control Model for Inter-System Cross-Domain in Multi-Domain Environment

Yunliang Li [1,*,†], Zhiqiang Du [2,*,†], Yanfang Fu [2,*] and Liangxin Liu [2]

[1] School of Armament Science and Technology, Xi'an Technological University, Xi'an 710021, China
[2] School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710021, China
* Correspondence: lyl@st.xatu.edu.cn (Y.L.); duzhiqiang@xatu.edu.cn (Z.D.); fuyanfang@xatu.edu.cn (Y.F.)
† These authors contributed equally to this work.

Abstract: Information service platforms or management information systems of various institutions or sectors of enterprises are gradually interconnected to form a multi-domain environment. A multi-domain environment is convenient for managers to supervise and manage systems, and for users to access data across domains and systems. However, given the complex multi-domain environment and many users, the traditional or enhanced role-based access control (RBAC) model still faces some challenges. It is necessary to address issues such as role naming conflicts, platform–domain management conflicts, inter-domain management conflicts, and cross-domain sharing difficulties. For the above problems, a role-based access control model for inter-system cross-domain in multi-domain environment (RBAC-IC) is proposed. This paper formally defines the model, divides roles into abstract roles and specific roles, and designs the operating process of the access control model. The model has four characteristics: support role name repetition, platform–domain isolation management, inter-domain isolation management, and fine-grained cross-domain sharing. By establishing security violation formulas for security analysis, it is finally shown that RBAC-IC can operate safely.

Keywords: role-based access control (RBAC); inter-system cross-domain; access control; multi-domain environment; information service platform

check for updates

Citation: Li, Y.; Du, Z.; Fu, Y.; Liu, L. Role-Based Access Control Model for Inter-System Cross-Domain in Multi-Domain Environment. *Appl. Sci.* **2022**, *12*, 13036. https://doi.org/10.3390/app122413036

Academic Editors: Christina Thorpe and Stephen O' Shaughnessy

Received: 8 October 2022
Accepted: 13 December 2022
Published: 19 December 2022

1. Introduction

In recent years, more and more firms or organizations have integrated the previously independent management information systems into a comprehensive information service platform [1–3]. In addition, cloud computing, distribution and other information technologies are also developing rapidly. SaaS [4,5], blockchain [6,7] and other methods interconnect the information service platforms of various sectors within the enterprise or various institutions within the organization, forming a multi-domain environment. In addition, A multi-domain environment accommodates various information systems deployed in different domain servers, and the structure is composed of three layers: platform in multi-domain environment, domain, and system. Figure 1 shows the hierarchical structure of the multi-domain environment. For example, the medical alliance is a multi-domain environment, and each hospital in the alliance belongs to a different domain. Each domain contains various management information systems owned by the hospital. The permissions of the same functional staff in different hospitals (domains) are different, and the data need to be isolated. Permissions of the same personnel in different hospitals (domains) are different, and the data need to be isolated. However, the medical alliance has business needs such as cross-hospital collaborative treatment and prescription circulation, so medical staff from different hospitals need to access data across domains.

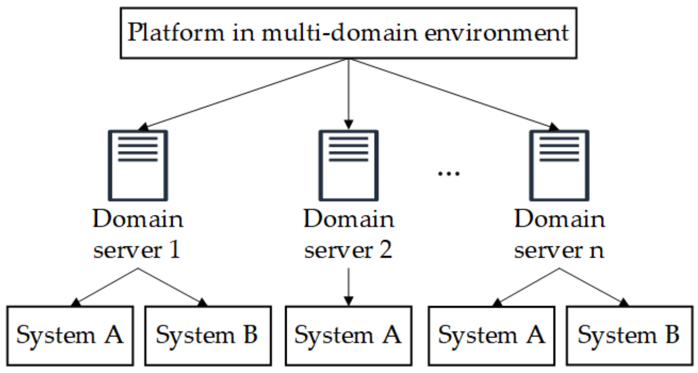

Figure 1. Hierarchical structure of multi-domain environment.

In such a multi-domain environment, it is conducive to data sharing among enterprise sectors or various institutions within the organization, and it is also beneficial for managers to supervise and manage the systems. Access control can prohibit illegal users from accessing resources in the system and prevent unauthorized operations by legitimate users in the system. There are many access control technologies, among which role-based access control (RBAC) [8] introduces the concept of role between users and permissions. It assigns permissions to roles based on business responsibilities, and then grants roles to users. The introduction of "role" simplifies the management of access control in complex systems, and RBAC has been widely used [9–12]. However, multi-domain environment is necessary to ensure data isolation among systems and to satisfy cross-domain sharing of data by users. Under these premises, how to improve RBAC and design an access control model for inter-system cross-domain, to control of user access is an urgent problem to be solved.

At present, there is a lot of research on the cross-domain access control model. However, there are still the following problems to be solved in the current research results:

- Role naming conflict: After the introduction of the role concept [8], although the same roles have similar responsibilities in different domains, it is necessary to allow the same roles to have different permissions in different domains.
- Platform–domain management conflict: Avoid domain administrators from unauthorized manipulation of the platform, and platform administrators from accessing domain server business data, resulting in privacy leakage.
- Inter-domain management conflicts: In a multi-domain environment, different domains need to be isolated for management, and data among systems also need to be isolated.
- Fine-grained cross-domain sharing: Meet users' needs for inter-system cross-domain access control. When a role is authorized cross-domain, only the required users are authorized, and the role authorization rights are not assigned to other domain administrators.

Aiming at the above problems, a role-based inter-system cross-control model (RBAC-IC) is designed. Based on the traditional RBAC model, roles are extended to abstract roles and specific roles. Abstract roles define the scope of responsibility, set hierarchies and constraints of roles; specific roles are responsible for assigning permissions and authorizing users. Platform personnel are divided into three categories: platform administrators, domain administrators and ordinary users. The two types of administrators are managed at the levels of platform and domain, respectively. Ordinary users can apply for authorized specific roles inner-domain or cross-domain, and access system resources with the authorized specific roles. The model has four characteristics: support role name repetition, platform–domain isolation management, inter-domain isolation management, and fine-grained cross-domain sharing.

The rest of this paper is organized as follows. Section 2 describes the research status in related fields. We introduce the proposed model by formal definition in Section 3. In Section 4, we expound on the execution flow after applying RBAC-IC on the multi-domain environment platform. Section 5 introduces how to apply RBAC-IC to the multi-domain information service platform. The features and security of RBAC-IC are analyzed in Section 6. It is concluded in Section 7.

2. Related Work

Sandhu et al. introduced the concepts of role hierarchy, constraint, etc., extended RBAC into four forms and collectively called it the RBAC96 model [13]; most of the later RBAC improvement schemes depended on the RBAC96 model. Later, Sandhu et al. proposed the ARBAC97 model [14], which divides the roles into mutually exclusive regular roles and administrative roles granted to the security administrators in the system; ARBAC97 standardizes and defines the operations of administrative roles to regular roles within the system. Ferraiolo et al. proposed a standard for RBAC [15], which defines some components of RBAC and their semantics, and regulates the operation and management of RBAC; The standard plays a normative role for subsequent RBAC research work. However, when solving inter-system cross-domain access control problems, if these traditional RBAC models are used, it can lead to a situation of role explosion [16], resulting in many redundant roles with similar permissions and easy confusion.

There have been many studies on improving the RBAC model to solve the shortcomings of the traditional RBAC model. Uddin et al. proposed a dynamic access control model AW-TRBAC [17], which assigns tasks according to users' roles, and access permissions are only available when tasks are executed in the workflow. The dynamic nature of the model alleviates the role explosion problem, but the security administrator does not define the management scope in the model and cannot resolve the inter-domain management conflict. In references [18–20], the concept of multi-dimensional roles is proposed to isolate tenants. Among them, reference [20] uses 4D-Role with user categories, and user categories include tenant users and platform users, making cloud platform management independent of tenant management. Strictly isolating users among different tenants is beneficial to privacy protection, but it is not conducive to information sharing among tenants.

Freudenthal et al. proposed a distributed role-based access control (dRBAC) [21]. In dRBAC, each entity uses its own name as the namespace of its publishing role, and the entity can delegate the role assignment rights to other entities, which in turn reduces role naming conflicts and permission sharing. However, these assignment rights can be passed so continually that it cannot control its indirect delegation. Tang et al. [22] proposed a multi-tenant RBAC model for collaborative cloud services. The issuer of the truster establishes a trust relation with the trustee, and the users in the trustee can authorize the roles in the truster to complete multi-tenant data sharing and isolation. The issuer needs to rely on each trustee to establish public role sets, and the truster cannot control the trustee's user authorization. Abdelfattah et al. [23] used the role-to-role (RTR) mapping rules to map a role in the organization with other organizational roles through the proposed role mapping algorithm, so that users can share other organizational resources. References [24–26], respectively, use inter-domain role mapping (IDRM) and role cross-domain inheritance to solve the problem of cross-domain sharing. References [23–26] have the same defects as in reference [22]. The truster domain cannot control how the trustee domain administrator authorizes roles to users, and the grain is coarse.

Uikey et al. proposed an RBAC architecture for multi-domain cloud environment [27]. Service providers and the domain administrators are, respectively, responsible for access control and access control policy management. There is a certain isolation between the service provider and the domain, but the service provider is allowed to modify the domain policies. Following that modification, the domain administrator needs to review and redefine the policy, which increases the management burden of the domain administrator. In addition, when users request cross-domain access, the domain administrator sends the policy to the corresponding domain, without considering the policy differences between different domains. There are also some studies that determine whether to authorize roles to users based on attributes [28] or points [29]. These concepts make the granularity of RBAC model finer, but the computing process has to be guaranteed to be reasonably reliable, otherwise the security of the access control process will be affected.

3. Proposed Model

3.1. Overview

In order to solve the problems of role naming conflicts, inter-domain management conflicts and cross-domain sharing difficulties. RBAC-IC divides roles into abstract roles and specific roles. Abstract roles do not need to assign permissions and authorize users. It is to define the responsibilities of positions contained in the system, set inheritance relationships and constraints among roles, and form the mapping various jobs or positions to system roles. Specific roles are an instantiation of an abstract role, and its purpose is to grant it to users to obtain the appropriate permissions. The specific role needs to be associated with an abstract role, and assign corresponding permissions to it according to the requirements. At the same time, the specific role inherits the role inheritance and constraints of the associated abstract role. Thus, RBAC-IC has the characteristics of support role name repetition, inter-domain isolation management and fine-grained cross-domain sharing.

In order to solve the problem of platform–domain management conflict, RBAC-IC divides platform personnel into three categories: platform administrators, domain administrators and ordinary users (users). The platform administrator is responsible for management operations for platform level such as platform configuration, system development and configuration (creating abstract roles, creating permissions, etc.), and deploying systems for domain servers. The domain administrator is responsible for creating specific roles of the deployed system, assigning permissions to specific roles, and authorizing ordinary users. At the same time, when users in a local domain apply for cross-domain access, the local domain administrator is responsible for sending cross-domain authorization requests to the application domain administrator. Ordinary users are users who use the application system. After granting specific roles, they can execute permissions by establishing sessions. The sessions activate a subset of the specific roles that ordinary users have. The permissions available to ordinary users are the aggregate of the permissions of all roles in activated sessions. Ordinary users need permission discrimination to access resources. After passing the discrimination, they can read or manipulate system data. Thus, RBAC-IC has the characteristics of platform–domain isolation management.

A role-based access control model for inter-system cross-domain in multi-domain environment (RBAC-IC) is defined in Figure 2.

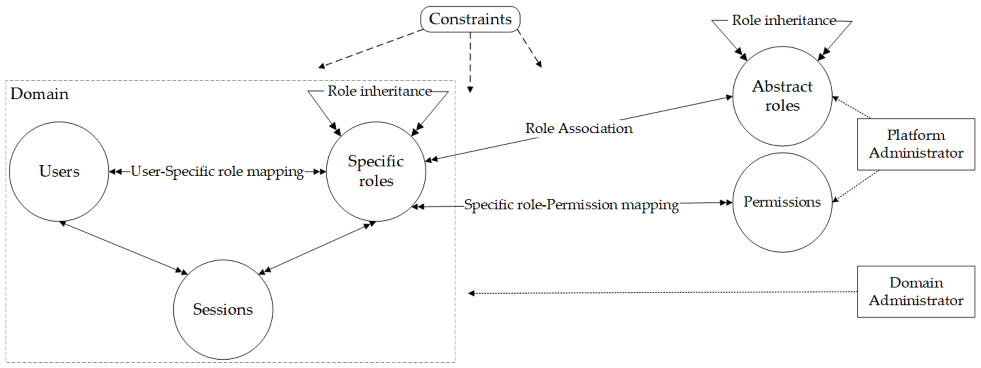

Figure 2. RBAC-IC sketch map.

3.2. Formal Definition of Model Sets

***U_d* (*Users*).** Set up a user set for each domain and the elements are composed of *user*, *user* is a subject in access control, which can be authorized specific roles and activate sessions, usually a person, device, or process, etc.

A *user* is two-tuples formed as <*category*, *domain*>, where *category* includes platform administrators, domain administrators and ordinary users; *domain* is used to identify the domain of domain administrators and ordinary users; *domain* of platform administrators is empty. Formally:

$$U_d = \{user_1, user_2, \ldots, user_n\}\ (d = 1, 2, \ldots, m) \tag{1}$$

is the set of all user in domain d; m is the number of domains in the platform; n is the number of users in domain d.

***P* (*Permissions*).** A permission set is set for the entire platform, and the elements are composed of *permission*, *permission* is the authorization of the subject to perform some operation on the object.

The *permission* is triple-tuples formed as <*category*, *operation*, *system*>, where *category* identifies the type of object manipulated by the permission, *operation* is a way of operating a *category*, such as read, write or executable; *system* indicates that the *permission* is valid in the *system*. Formally:

$$P = \{permission_1, permission_2, \ldots, permission_{\mathrm{n}}\} \tag{2}$$

is the set of all permission in the platform; n is the number of permissions in the platform.

O_d (*Objects*). Set up an object set for each domain and the elements are composed of *object*. The *object* is the object (resources) in the system and be manipulated by users, which usually exist in the form of files, data, etc.

An *object* is triple-tuples formed as <*category*, *domain*, *system*>, where *category* identifies the type of object; *domain* and *system* indicate that the *object* is stored in the *system* of the *domain*. Formally:

$$O_d = \{object_1, object_2, \ldots, object_n\}\ (d = 1, 2, \ldots, m) \tag{3}$$

is the set of all object in domain d; m is the number of domains in the platform; n is the number of objects in domain d.

***AR* (*Abstract_Roles*).** An abstract set is set for the entire platform and the elements are composed of *abstract_role*. The *abstract_role* is abstraction of a set of responsibilities within a system of the platform. It cannot be authorized to users, nor can abstract roles be assigned permissions.

An *abstract_role* is triple-tuples formed as <*system*, *primary_role*, *constraint*>, where *system* indicates that the *abstract_role* is valid in the *system*; *primary_role* is the set of abstract roles which it inherits; *constraint* is the constraint set of the abstract role. Formally:

$$AR = \{abstract\_role_1, abstract\_role_2, \ldots, abstract\_role_n\} \tag{4}$$

is the set of all abstract role in the platform; n is the number of abstract roles in the platform.

SR_d (*Specific_Roles*). Set up a specific role set for each domain and the elements are composed of *specific_role*. The *specific_role* is assigned a set of permissions according to the responsibilities of the associated abstract role and the personalized requirements of the domain, and authorized the users to complete a certain business.

A *specific_role* is five-tuples formed as <*abstract_role*, *permissions*, *domain*, *system*, *valid_time*>, where *abstract_role* is the abstract role associated with the specific role, *permissions* are a group of permission assigned to the specific role, *domain* and *system* indicate that the *specific_role* is valid in the *system* of the *domain*, and *valid_time* is the valid time of the specific role, which is generally set when users apply for cross-domain access. When $valid\_time \to +\infty$, it means that the specific role is permanently valid. Specific roles can only be used in the designated system within its set domain, and must be valid for a period. Formally:

$$SR_d = \{specific\_role_1, specific\_role_2, \ldots, specific\_role_n\}\ (d = 1, 2, \ldots, m) \tag{5}$$

is the set of all specific role in domain d; m is the number of domains in the platform; n is the number of specific roles in domain d.

***S_d* (*Sessions*).** Set up a session set for each domain and the elements are composed of session. When the *user* performs tasks, the *session* is the mapping between the user and the specific roles that need to be activated.

A *session* is two-tuples formed as <*user*, *specific_role*>, where *user* and *specific_role* represent the user and specific roles are activated by the user, respectively. Formally:

$$S_d = \{session_1, session_2, \ldots, session_n\}\ (d = 1, 2, \ldots, m) \tag{6}$$

is the set of all session in domain d; m is the number of domains in the platform; n is the number of sessions in domain d.

3.3. Formal Definition of Model Relationships

$USR_d \subseteq U_d \times SR_d$. denote a set of many-to-many relationships from users to specific roles. Formally:

$$\forall(user_i, specific\_role_j) \in USR_d\ (user_i \in U_d, specific\_role_j \in SR_d)\ (d = 1, 2, \ldots, l; i = 1, 2, \ldots, m; j = 1, 2, \ldots, n) \tag{7}$$

$SRP_d \subseteq SR_d \times P_d$. denote a set of many-to-many relationships from specific roles to permissions. Domain administrators need to assign permissions to specific roles according to the principle of least privilege [30] and the associated abstract roles. The user needs to access the object through specific roles that conforms to the principle of minimum authority, and cannot directly access the object by bypassing the specific role, nor can they directly assign the permission to the user. Formally:

$$\forall(specific\_role_i, permission_j) \in SRP_d\ (specific\_role_i \in SR_d, permission_j \in P_d)\ (d = 1, 2, \ldots, l; i = 1, 2, \ldots, m; j = 1, 2, \ldots, n) \tag{8}$$

$US_d \subseteq U_d \times S_d$. denote a set of one-to-many relationships from a user to sessions. Formally:

$$\forall(user, session_i) \in US_d\ (user \in U_d, session_i \in S_d)\ (d = 1, 2, \ldots, m; i = 1, 2, \ldots, n) \tag{9}$$

$SRAR_d \subseteq SR_d \times AR$. denote a many-to-one relationship set from specific roles to an abstract role. A specific role can only be associated with one abstract role, and an abstract role can be associated with multiple specific roles. Formally:

$$\forall(specific\_role_i, abstract\_role) \in SRAR_d\ (specific\_role_i \in SR_d, abstract\_role \in AR)\ (d = 1, 2, \ldots, m; i = 1, 2, \ldots, n) \tag{10}$$

$sr\_association(specific\_role \in SR_d) \rightarrow AR$. denote a mapping from a specific role to an abstract role. Formally:

$$sr\_association(specific\_role \in SR_d) = \{abstract\_role \in AR \mid (specific\_role, abstract\_role) \in SRAR_d\} \tag{11}$$

Property 1. *A specific role has one and only one associated abstract role.*

Proof of Property 1. The formal proof is as follows:
Suppose,

$$\forall specific\_role_1 \in SR_1;$$

$$\forall abstract\_role_1, abstract\_role_2 \in AR;$$

If,

$$sr\_association(specific\_role_1) = abstract\_role_1;$$

$$sr\_association(specific\_role_1) = abstract\_role_2;$$

Because,

$$sr\_association(specific\_role \in SR_d) \rightarrow AR;$$

So,

$$abstract\_role_1 = abstract\_role_2.$$

□

3.4. Formal Definition of Model Hierarchies

$\boldsymbol{ARH \subseteq AR \times AR}$**.** denote a set of inheritance relationships among abstract roles, which is an antisymmetric partial order relationship. Formally:

$$(abstract\_role' \succeq abstract\_role) \in ARH\ (abstract\_role', abstract\_role \in AR) \tag{12}$$

where *abstract_role′* is called the senior abstract role of *abstract_role*, and *abstract_role* is called the primary abstract role of *abstract_role′*.

$\boldsymbol{SRH_d \subseteq SR_d \times SR_d}$**.** denote a set of inheritance relationships among specific roles, which is an antisymmetric partial order relationship. Formally:

$$(specific\_role' \succeq specific\_role) \in SRH_d\ (specific\_role', specific\_role \in SR_d)\ (d = 1, 2, \ldots, n) \tag{13}$$

where *specific_role′* is called the senior specific role of *specific_role*, and *specific_role* is called the primary specific role of *specific_role′*. In addition, *specific_role′* inherits all the permissions of *specific_role*.

Property 2. *The inheritance relationship among specific roles is consistent with the inheritance relationship among the associated abstract roles. The formal expression is as follows,*
Suppose,

$$\forall specific\_role_1, specific\_role_2 \in SR_1;$$

$$\forall abstract\_role_1, abstract\_role_2 \in AR;$$

If,

$$sr\_association(specific\_role_1) = abstract\_role_1;$$

$$sr\_association(specific\_role_1) = abstract\_role_2;$$

$$abstract\_role_1 \succeq abstract\_role_2;$$

So,

$$specific\_role_1 \succeq specific\_role_2.$$

Example 1. *There are four abstract roles A, B, C and D in the system. B and C inherit A; D inherits B and C; A has a constraint. Create four specific roles A′, B′, C′, and D′ to be associated with A, B, C, and D, respectively. Then, the inheritance relationship between the four specific roles is B′ and C′ inherit A′; D′ inherits B′ and C′; A′ inherits constraints of A. There are no other inheritance relationships, as shown in Figure 3.*

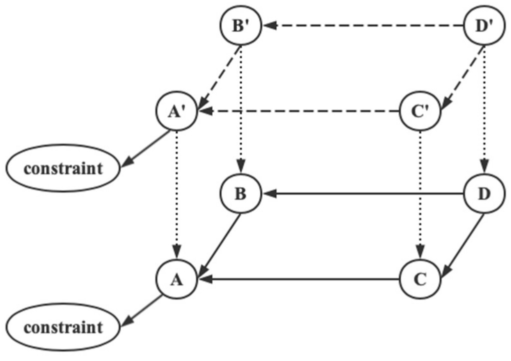

Figure 3. Example diagram of role inheritance.

$\boldsymbol{ar\_primary\_role(abstract\_role \in AR) \to 2^{AR}}$**.** denote all abstract roles inherited by an abstract role. Formally:

$$ar\_primary\_role(abstract\_role \in AR) = \{AR_i \mid AR_i \subseteq AR\} \tag{14}$$

Property 3. *If the abstract role has multiple inheritance, the abstract role set it inherits includes the directly inherited abstract role set, and the abstract role set that it inherits indirectly. The formal expression is as follows,*
Suppose,

$$\forall abstract\_role', AR_{a\cup b}, AR_a, AR_b \subseteq AR;$$

If,

$$AR_a \text{ denote the set of direct primary abstract roles of } abstract\_role';$$

$$AR_b \text{ denote the full set of indirect primary abstract roles for } abstract\_role';$$

$$AR_b = \bigcup_{abstract\_role' \succeq abstract\_role} ar\_primary\_role(abstract\_role)$$

So, all abstract roles inherited by abstract_role′ are,

$$AR_{a\cup b} = ar\_primary\_role(abstract\_role') = AR_a \cup AR_b.$$

***sr_primary_role*($\boldsymbol{specific\_role \in SR_d}$)** $\rightarrow 2^{SR_d}$. *denote all specific roles inherited by a specific role. Formally:*

$$sr\_primary\_role(specific\_role \in SR_d) = \left\{SR_{d_i} \;\middle|\; SR_{d_i} \subseteq SR_d\right\} \tag{15}$$

Property 4. *If a specific role has multiple layers of inheritance, the set of specific roles it inherits includes the set of directly inherited specific roles and the set of specific roles that it inherits indirectly. The formal expression is as follows,*
Suppose,

$$\forall specific\_role',\ SR_{1_{a\cup b}},\ SR_{1_a},\ SR_{1_b} \subseteq SR_1$$

If,

$$SR_{1_a} \text{ denote the set of direct primary specific roles of } specific\_role';$$

$$SR_{1_a} \text{ denote the full set of indirect primary specific roles for } specific\_role';$$

$$SR_{1_b} = \bigcup_{specific\_role' \succeq specific\_role} sr\_primary\_role(specific\_role)$$

So, all specific roles inherited by specific_role′ are,

$$SR_{1_{a\cup b}} = sr\_primary\_role(specific\_role') = SR_{1_a} \cup SR_{1_b}$$

3.5. Formal Definition of Model Functions

***user_authorization*($\boldsymbol{user \in U_d}$) $\boldsymbol{\longrightarrow 2^{SR_e}}$.** denote all specific roles authorized to a user. Formally:

$$userauthorization(user \in U_d) = \{SR_{e_i} \mid SR_{e_i} \subseteq SR_{e_i},\ (user,\ SR_{e_i}) \in USR_d\}\ (e = 1, 2, \ldots, n) \tag{16}$$

when $e = d$, SR_{e_i} indicates the specific role set of local domain for the *user*; when $SR_{e_i} = \emptyset$, it explains that the *user* does not have any specific role of domain e.

***sr_assignment*($\boldsymbol{specific\_role \in SR_d}$) $\boldsymbol{\longrightarrow 2^{P}}$.** denote all the permissions assigned to a specific role. Formally:

$$sr\_assignment(specific\_role \in SR_d) = \{P_i \mid P_i \subseteq P,\ (specific\_role,\ P_i) \in SRP_d\} \tag{17}$$

Property 5. *If a specific role has a primary specific role, its permission set includes the directly assigned permissions and the permissions of all its primary roles. The formal expression is as follows: Suppose,*

$$\forall specific\_role', specific\_role \in SR_1;$$

$$\forall P_{a\cup b}, P_a, P_b \subseteq P;$$

If,

P_a denotes a permission set directly assigned to *specific_role'*;

P_b denote the permission set of all primary specific roles of *specific_role'*,

$$P_b = \bigcup_{specific\_role \in sr\_inheritance(specific\_role')} sr\_assignment(specific\_role)$$

So, all permissions assigned by specific_role' are,

$$P_{a\cup b} = sr\_assignment(specific\_role') = P_a \cup P_b.$$

$\boldsymbol{user\_assignment(user \in U_d) \rightarrow 2^P}$. *denote all permissions a user has. Formally:*

$$user\_assignment(user \in U_d) = \bigcup_{specific\_role \in user\_authorization(user)} sr\_assignment(specific\_role) \tag{18}$$

$\boldsymbol{user\_sessions(user \in U_d) \rightarrow 2^{S_e}}$. *denote all sessions activated by a user. Formally:*

$$user\_sessions(user \in U_d) = \{S_{d_i} \mid S_{d_i} \subseteq SR_d, (user, S_{d_i}) \in US_d\} \tag{19}$$

$\boldsymbol{session\_user(session \in S_d) \rightarrow user \in U_d}$. *denote from a session to a user mapping. Formally:*

$$session\_user(session \in S_d) = \{user \mid user \in U_d, (user, session) \in US_d\} \tag{20}$$

$\boldsymbol{session\_sr(session \in S_d) \rightarrow 2^{SR_e}}$. *denote all specific roles activated by a session. Formally:*

$$session\_sr(session \in S_d) = \{SR_{e_i} \mid SR_{e_i} \subseteq SR_d, (session\_user(session), SR_{e_i}) \in USR_d\}\ (e = 1, 2, \ldots, n) \tag{21}$$

when $e = d$, SR_{e_i} indicates the specific role set of the local domain activated for the session; when$SR_{e_i} = \varnothing$, it explains that the session does not activate any specific role of domain e.

$\boldsymbol{session\_permission(session \in S_d) \rightarrow 2^P}$. *denote all permissions activated by a session. Formally:*

$$sessionpermission(session \in S_d) = \bigcup_{specific\_role \in session\_sr(session)} sr\_assignment(specific\_role) \tag{22}$$

4. RBAC-IC Execution

The workflow diagram of RBAC-IC is shown in Figure 4.

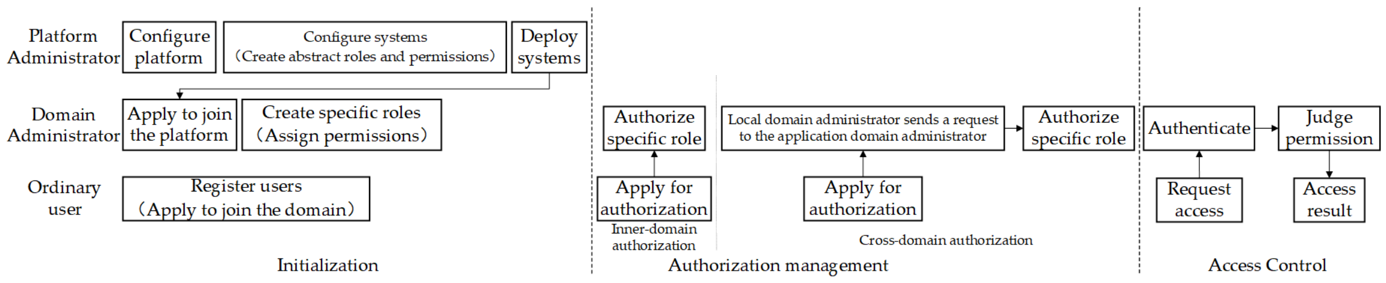
Platform Administrator
Configure platform
Configure systems (Create abstract roles and permissions)
Deploy systems
Domain Administrator
Apply to join the platform
Create specific roles (Assign permissions)
Ordinary user
Register users (Apply to join the domain)
Initialization
Authorize specific role
Apply for authorization
Inner-domain authorization
Local domain administrator sends a request to the application domain administrator
Authorize specific role
Apply for authorization
Cross-domain authorization
Authorization management
Authenticate
Judge permission
Request access
Access result
Access Control

Figure 4. Workflow of RBAC-IC.

4.1. Initialization Work

When developing and configuring the system, the platform administrator needs to create abstract roles for domain administrators to create specific roles. The platform administrator uploads the name, *system*, *primary_role* and *constraint* of the abstract role to the platform. The platform checks whether name of the abstract role exists and whether the *system* is empty. After passing all checks, the platform database stores the abstract role information and inheritance relationship into the abstract role set and inheritance relationship set. Finally, the platform returns the execution result to the platform administrator. Formally:

$$\begin{gathered} AR = AR \cup abstract\_role{<}system, primary\_role, constraint{>}; \\ ARH = ARH \cup (abstract\_role, abstract\_role[primary\_role]) \end{gathered} \tag{23}$$

Platform administrators also need to create permissions for domain administrators to assign permissions to specific roles. When creating a permission, the platform administrator uploads the name, *category*, *operation* and *system* of the permission to the platform. The platform checks whether the name of the permission exists and whether other information is empty. After passing all checks, the platform database stores the permission information into the permission set. Finally, the platform returns the execution results to the platform administrator. Formally:

$$P = P \cup permission{<}category, operation, system{>} \tag{24}$$

After deploying the system for the domain server, the domain administrator needs to create specific roles for the deployed system. Specific roles are used to authorize the user to complete the access operation. The domain administrator also needs to create specific roles with timeliness to meet users' needs for cross-domain sharing. When creating a specific role, the domain administrator uploads the name, *abstract_role*, *permissions*, *system* and *valid_time* to domain server. The domain server first checks whether the name of the specific role exists, then checks whether the *system* is empty, and then checks whether the associated abstract role and each assigned permission exist. If all checks pass, the domain of the domain administrator will be taken as the effective range (*domain*) of the specific role. The domain database stores the specific role information and associated relationships into a corresponding set. Finally, the execution results will be returned to the domain administrator. Formally:

$$\begin{gathered} SR_d = SR_d \cup specific\_role{<}abstract\_role, permissions, domain, system, valid\_time{>}; \\ SRAR_d = SRAR_d \cup (specific\_role, specific\_role[abstract\_role]); \\ SRP_d = SRP_d \cup (specific\_role, specific\_role[permissions]); \\ SRH_d = SRH_d \cup (specific\_role, specific\_role[primary\_role]) \end{gathered} \tag{25}$$

After the deployment of the domain server is completed, ordinary users can register to join the domain. Formally:

$$U_d = U_d \cup user{<}ordinary\_user, domain{>} \tag{26}$$

4.2. Authorization and Access Control

The authorization and access control framework of RBAC-IC is shown in Figure 5. Figure 5 shows that the domain administrators of domain A and domain B have created a specific role associated with the same abstract role in their respective domains, and assigned different permissions. An ordinary user in domain A wants to access an object in domain B, and needs to authorize a relevant specific role in domain B. The user first applies to the administrator of the local domain, and the administrator sends a notification to domain B. If the administrator of domain B agrees, the specific role will be authorized to the user. After the user obtains the specific role, he can access resources in domain B across domains. Sections 4.2.1 and 4.2.2 describe the details of authorization and access control processing.

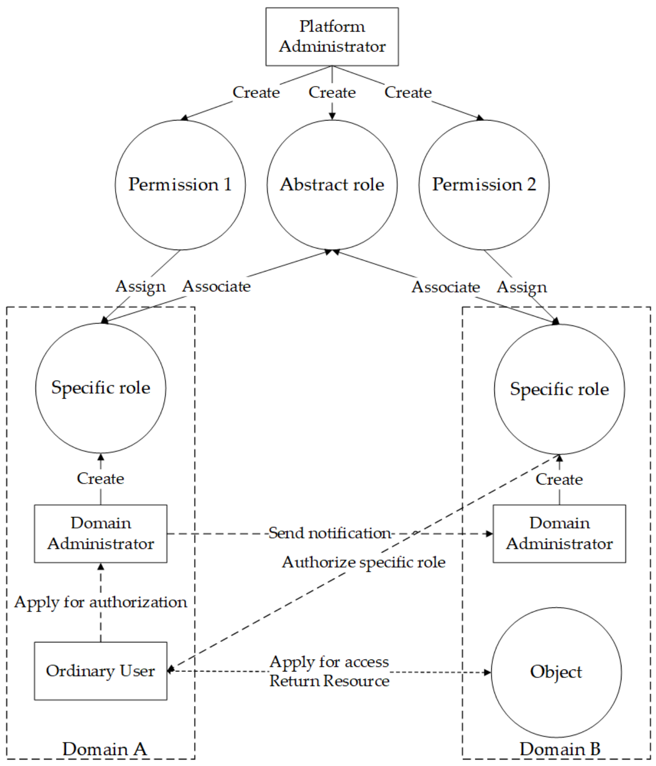

Figure 5. Authorization and access control framework of RBAC-IC.

4.2.1. Authorization Management

Users who want to access the objects in the system need to have corresponding permissions, which are obtained by authorizing specific roles. For inner-domain authorization, the user applies for a specific role to the domain server where he belongs. If the domain administrator agrees to authorize, the domain server will check the specific role and authenticate. After the authentication is passed, the domain server will check whether the user meets the constraint required by the specific role. After the inspection is correct, the domain server will authorize the specific role to the user. The sequence diagram of applying for inner-domain authorization is shown in Figure 6.

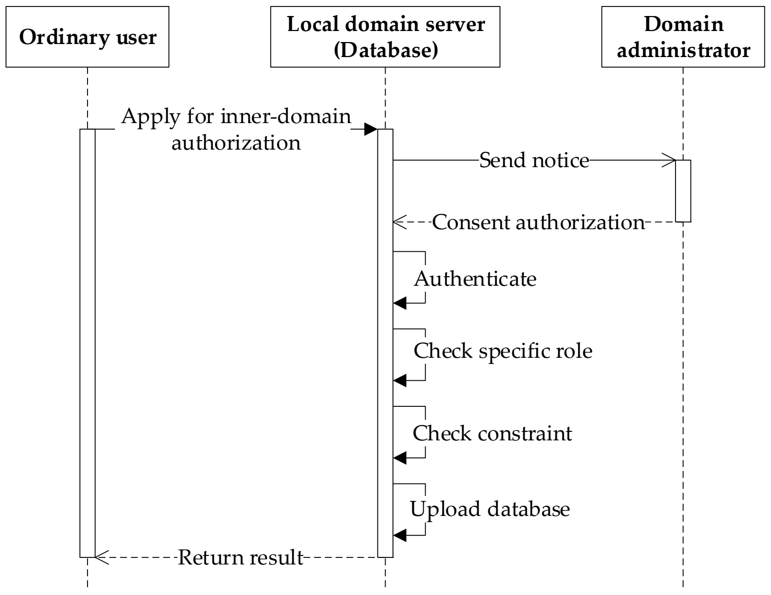

Figure 6. Sequence diagram of applying for inner-domain authorization.

When users need to cross-domain accesses objects, it is necessary to apply for cross-domain authorization, and the domain administrator of the requested domain authorizes the specific role. The user applies for cross-domain authorization to the local domain server. After the authentication is passed and the local domain administrator agrees, the local domain server sends a request to the corresponding domain server. If the administrator of the requested domain also agrees to authorize, the requested domain server checks the specific role and determines whether the user meets the constraint. After the inspection is correct, the domain administrator authorizes the specific role to the user. In addition, domain administrators can set a valid time of the specific role. When valid time is reached, the specific role will become invalid, and the user will not be able to continue to access corresponding resources in the domain. The sequence diagram of applying for cross-domain authorization is shown in Figure 7.

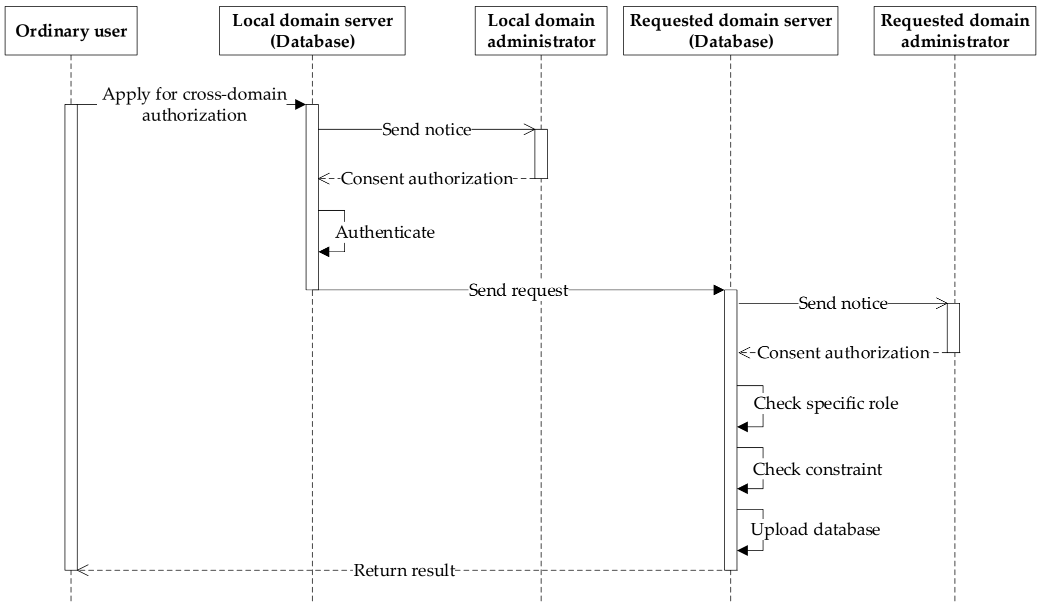

Figure 7. Sequence diagram of applying for cross-domain authorization.

4.2.2. Access Control

When the user requests to access objects of system, the system needs to control the user's behavior and decide whether to allow the user to access the objects through authentication and authority judgement. The access control can be automatically determined by the domain server without the operation of the administrator, as shown in Algorithm 1. The process of cross-domain access control is like that of inner-domain access control, but it is different when applying for authorization. After the user applies for access to the object, the user is authenticated in the user's domain. Then, the permission discrimination is carried out in the domain of the user's target object. The permission discrimination process checks the correctness of the object, specific role, and permission in turn. Then, the permission discrimination is carried out in the domain of the user's target object. It includes checking whether the object exists, whether the user has the declared specific role and permission, whether the specific role and permission declared by the user are valid in the applied domain and system, and whether the specific role is in valid time. After checking that everything is correct, the user's operation behavior is executed.

Algorithm 1 Access control

```
Input: u(user), o(object), p(permission), sr(specific_role)
Output: bool
1:  Domain Server ← Ordinary User(u, o, p, sr)
2:  if $U_d[u]$ != true | | u[category] != ordinary_user then
3:      return false
4:  if $O_d[o]$ != true | | sr[domain] != o[domain] | | p[system] != o[system] | | p[category] != o[category] then
5:      return false
6:  for i = 0 to user_authorization(u).length - 1 do
7:      if user_authorization(u)[i] == sr && user_authorization(u)[i][valid_time][min_time] < current_time &&
    user_authorization(u)[i][valid_time][max_time] > current_time then
8:          for i = 0 to sr_assignment(sr).length - 1 do
9:              if sr_assignment(sr)[i] == permission then
10:                 $S_d = S_d \cup (u, sr)$
11:                 return true
12: return false
```

5. Case Analysis

In order to better understand and verify the RBAC-IC model, a set of multi-domain information service platform is developed, which uses the RBAC-IC model as the access control model. At present, the platform has been applied in Taishan Zhizhen Packaging. The main business of the group is to produce metal cans.

5.1. Platform Architecture

Taishan Zhizhen Packaging has several production subsidiaries in different cities. Each manufacturing subsidiary of this group has a domain server as a domain in the platform, called the production domain. Some parts or processes of packaging cans (such as can cover manufacturing, metal can printing, etc.) are processed by cooperative outsourcing companies, and the domain server of the outsourcing company is called the outsourced domain. All administrative departments of the group are located at the group headquarters, and they share a domain server called the administrative domain. Each production and outsourced domain contain a production management information system to manage production data, and a financial management information system to manage the company's finances. The administrative domain contains a sales management information system, an HR management information system, and a financial management information system to, respectively, manage the corresponding business data. The multi-domain environmental structure of Taishan Zhizhen Packaging is shown in Figure 8. Clearly, the business logic has been suitably simplified by considering only a production domain, an outsourced domain and an administrative domain, and three management information systems: production, sales and finance.

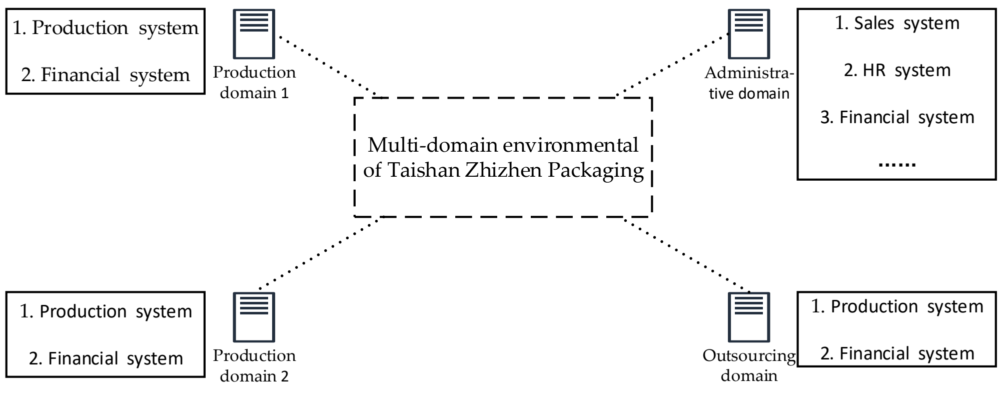

Figure 8. Multi-domain environmental structure of Taishan Zhizhen Packaging.

The back-end business and API interfaces of the platform and system are developed using Spring Boot 2.7.0. MySQL 5.7 is used for platform database and domain database. The data volume and concurrency of the production system in the production domain and outsourcing domain are high, and users will interact with the production system uninterruptedly during the production process. Therefore, it is deployed on a high-performance cloud server, and the CPU is Intel Xeon (Cascade Lake) Platinum 8269CY@2.50 GHz (64 G memory). The administrative domain is deployed on the server, and the CPU of the server is Intel Xeon E5-2620 v3@2.40 GHz (32 G memory).

5.2. Design of Platform Access Control Model

In the platform, platform administrators are responsible for the design of abstract roles and permissions, while domain administrators are responsible for specific role design, user registration, and user authorization. The responsibilities of the two types of administrators do not intersect, reflecting the platform–domain isolation management feature of the RBAC-IC and solving the platform–domain management conflict problem.

5.2.1. Design of Abstract Roles

Abstract roles are designed as shown in Table 1. AR1 and AR3 are the primary roles of AR2 and AR4, respectively. Specific roles associated with AR2 inherit all the permissions of specific roles associated with AR1. The cardinality constraint limit for AR2, AR4 and AR7 is 1; that is, their associated specific roles can only be authorized to one user at most. AR2 and AR4 have prerequisite constraints. When authorizing a specific role associated with AR2 or AR4 to a user, the user must have authorized the specific role associated with AR1 or AR3. AR5 and AR6 are mutex, and their associated specific roles cannot be authorized to a user at the same time.

Table 1. Design of abstract roles.

Serial Number	Name	System	Primary Role	Constraint
AR1	Production staff	Production		
AR2	Production executive	Production	AR1	Cardinality (1) Prerequisite (AR1)
AR3	Sales staff	Sales		
AR4	Sales executive	Sales	AR3	Cardinality (1) Prerequisite (AR3)
AR5	Accountant	Finance		Mutex (AR6)
AR6	Auditor	Finance		Mutex (AR5)
AR7	Treasurer	Finance		Cardinality (1)

5.2.2. Design of Permissions

Permissions are designed as shown in Table 2. The permissions are valid within the specified systems of all domains.

5.2.3. Design of Specific Roles

Specific roles are designed as shown in Table 3. The subsidiary is responsible for the production of three-piece cans and two-piece cans. There are two specific roles SR1 and SR2 associated with AR1 in the production domain. SR1 and SR2 are assigned the read and write permissions for the data of three-piece cans and two-piece cans, respectively. The printing company in the outsourcing domain is responsible for printing metal materials for the three-piece cans. The printing company is only responsible for printing the three-piece cans. Therefore, only one specific role SR5 associated with AR1 is required, and SR5 is assigned read/write permissions for the data of two-piece cans. SR6, the production staff of the printing company, only has the permission to publish the production report of three-piece cans. In addition, the printing staff need to work according to the processing data of the three-piece cans. At this time, the domain administrator of the production domain needs to create the specific role SR4, and only assign the data read permission of the three-piece cans. SR4 is designed to provide cross-domain access for printing staff in outsourced domains.

Table 2. Design of permissions.

Serial Number	Category	Operation	System
P1	Data of three-piece cans	Input	Production
P2	Data of three-piece cans	Read	Production
P3	Data of two-piece cans	Input	Production
P4	Data of two-piece cans	Read	Production
P5	Production report of product A	Publish	Production
P6	Production report of product B	Publish	Production
P7	Sales data	Input	Sales
P8	Sales data	Read	Sales
P9	Sales report	Publish	Sales
P10	Financial statement	Publish	Finance
P11	Financial statement	Audit	Finance
P12	Financial report	Publish	Finance

Table 3. Design of specific roles.

Serial Number	Name	Abstract Role	Permission	Domain	System	Valid Time
SR1	Production staff of three-piece cans	AR1	P1, P2	Production	Production	+∞
SR2	Production staff of two-piece cans	AR1	P3, P4	Production	Production	+∞
SR3	Production Supervisor	AR2	P5, P6	Production	Production	∞
SR4	Outsourced printing staff	AR1	P1	Production	Production	2022-07-03T00:00:00Z 2022-07-05T23:59:59Z
SR5	Printing staff	AR1	P1, P2	Outsourced	Production	+∞
SR6	Production Executive	AR2	P5	Outsourced	Production	+∞
SR7	Sales staff	AR3	P7, P8	Administrative	Sales	+∞
SR8	Sales Executive	AR4	P9	Administrative	Sales	+∞
SR9	Accountant	AR5	P10	Administrative	Finance	+∞
SR10	Auditor	AR6	P11	Administrative	Finance	+∞
SR11	Treasurer	AR7	P12	Administrative	Finance	+∞

Due to the inheritance relationship, SR3 inherits all the permissions of SR1, SR2 and SR4, and SR6 inherits the permissions of SR5. Data and specific roles under different domain servers are isolated from each other, and the specific roles are valid in the designated systems and domains. Therefore, RBAC-IC has inter-domain isolation management of the RBAC-IC and solves the problem of inter-domain management conflict. SR3 and SR6 have the same name, but the two roles have different permissions in different domains, so they have the feature of support role name repetition, solving the problem of role naming conflicts. SR4 is set by the administrator for users in other domain to cross-domain access, and the valid time is set to 12 h.

5.2.4. User's Authorization

Design and authorization of users is shown in Table 4. RBAC-IC has three ways of authorizing the user a specific role. Users can be authorized multiple roles of a system in their domain, as shown in Figure 9a. For example, U1 can produce three-piece cans and two-piece cans, so it is authorized SR1 and SR2 of the production management information system. Users also can be authorized to have a specific role for multiple systems in their domain, as shown in Figure 9b. For example, U2 can be the sales executive and treasurer at the same time, so it is authorized SR8 and SR11 (assuming constraints are met). Users can be authorized specific roles in other domains, as shown in Figure 9c. For example, U3, the printing staff in the outsourcing domain, needs to carry out printing according to the data of three-piece cans in the production domain, and thus, it can be authorized SR4.

Table 4. User authorization.

User Number	Domain of User	Specific Role	Authorization Result
U1	Production	SR1	Allowance
U1	Production	SR2	Allowance
U2	Administrative	SR8	Allowance
U2	Administrative	SR11	Allowance
U3	Outsourced	SR4	Allowance
U1	Production	SR3	Allowance
U4	Production	SR3	Denial (Cardinality constraint)
U5	Administrative	SR8	Denial (Prerequisite constraint)
U6	Production	SR9	Allowance
U6	Production	SR10	Rejection (Static mutex constraint)

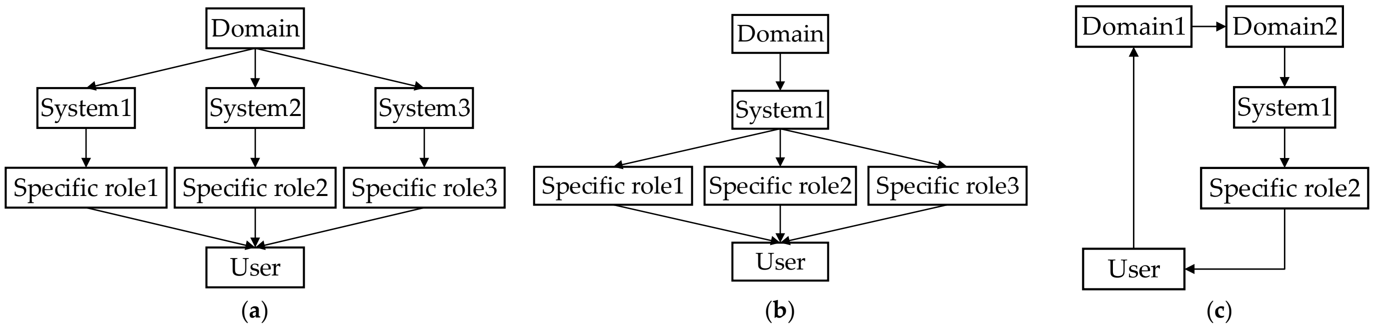

Figure 9. Diagram of the way users are authorized to have specific roles. (**a**) Single system multi-role authorize approach. (**b**) Multi-system multi-role authorize approach. (**c**) Cross-domain authorize method.

When authorizing specific roles to users, this will be limited by the constraint. After having already authorized SR3 to U1, authorizing SR3 to U4 will be denied because SR3 can only authorize one user at most. This is allowed when authorizing SR3 for U1, which already has the prerequisite role SR1 or SR2 required by SR3, but cannot authorize SR8 to U5 because U5 has not authorized the prerequisite role SR7 required by SR8, which violates prerequisite constraints. SR9 has been successfully authorized for U6, so continuing to authorize SR10 for U6 would be rejected because the two specific roles are mutex.

5.2.5. User's Access Control Operation

Suppose a user requests to input production data of three-piece cans into the production management information system in the production domain, the access control example is shown in Table 5. U7 does not exist in the production domain, so access is denied due to authentication failure. When U1 requests access using SR5, the domain of SR5 is inconsistent with that of the resource, and access is denied. When U1 requests access using P3, the resource type of P3 does not match the requested resource type, and access is denied. When U2 requests access using SR1, U2 does not authorize SR1, and access is denied. When U1 requests access using SR1 and P1, SR2 does not assign P1, and access is denied. When U1 requests access using SR1 and P1, at this point, authentication and permission discriminations all pass, and access is allowed. When U4 uses SR4 and P1 to request access, U4 performs authentication in the outsourced domain and determines the permission in the production domain. If they all pass, this cross-domain access is allowed.

Table 5. Example access control table.

User	Domain of User	Specific Role	Permission	Result of Access	Reason of Denial
U7	Production	SR1	P1	Denial	Authentication failed
U1	Production	SR5	P1	Denial	Mismatch between SR5 and resource
U1	Production	SR1	P3	Denial	Mismatch between P3 and resource
U2	Production	SR1	P1	Denial	U2 has not authorized SR1
U1	Production	SR2	P1	Denial	P1 is not assigned to SR2
U1	Production	SR1	P1	Allowance	/
U3	Outsourced	SR4	P1	Allowance	/

6. Model Evaluation

6.1. Model Characteristics

For support role name repetition, it is achieved by splitting roles into abstract and specific roles. Domain administrators create specific roles for systems in the domain, then assign permissions to specific roles based on responsibilities of the associated abstract roles. Different permissions are allowed for specific roles with the same name in different domains, so it will not cause role naming conflicts.

For platform–domain isolation management, RBAC-IC has designed three types of personnel: platform administrators, domain administrators and ordinary users. Platform administrators can create abstract roles, but abstract roles do not assign permissions and authorize users. Therefore, platform administrators cannot control the database of the domain server. Domain administrators can create specific roles that are valid only within their domain, and assign permissions to them and authorize them to ordinary users. In this way, privacy disclosure to platform administrators is avoided.

For inter-domain isolation management, RBAC-IC regards specific roles as a five-tuple, in which if domain and system element are different, the effective system and domain of the specific role are different. This ensures that systems and data in different domains are isolated from each other.

For fine-grained cross-domain sharing, domain administrators create special specific roles for cross-domain access, and users can cross-domain access after authorizing these specific roles. These specific roles are authorized by the domain administrator of the domain where they are located, and the authorizing rights of the roles are not assigned to other domain administrators, to achieve fine-grained authorization. In addition, domain administrators can also set valid time for these specific roles, and these specific roles will automatically become invalid after the expiration.

Table 6 shows the comparison between RBAC-IC and other schemes in the above four characteristics.

6.2. Security Analysis

Some security violation formulas are proposed to verify the security of the model. In any case, the security of the system can only be proved if these security violation formulas cannot be satisfied.

Table 6. Characteristics comparison between RBAC-IC and other schemes.

Schemes	Support Role Name Repetition	Platform–Domain Isolation Management	Inter-Domain Isolation Management	Cross-Domain Sharing
RBAC96 [13]	Not supported.	Not supported.	Not supported.	Not supported.
ARBAC97 [14]	Not supported.	Not supported. (Administrative role can be slightly refined to support.)	Not supported.	Not supported.
Uddin et al. [17]	Not supported. (Role redundancy can be reduced through highly dynamic workflow and task concepts.)	Not applicable.	Not applicable.	Not applicable.
Literature [18,19]	Yes. (Expand the role to two or three dimensions.)	Not supported.	Yes. (Roles are valid only within the specified scope.)	Not supported.
Zhang et al. [20]	Yes. (Expand the role to four dimensions.)	Yes. (The category element of the 4D role distinguishes the platform administrator, tenant administrator and user.)	Yes. (Roles are valid only within the specified scope.)	Not supported.
Ferraiolo et al. [21]	Yes. (Use the role namespace as the role prefix.)	Not applicable.	Yes. (It can be isolated through the role namespace.)	Fine-grained is not supported. (By delegate the role assignment rights to other entities.)
Tang et al. [23]	Not applicable.	Not applicable.	Fine-grained is not supported. (After the issuer of the truster establishes private role sets and public role sets, it establishes a trust relationship with the trustee. The issuer of the truster can authorize users roles of the public role set.)	
Uikey et al. [22]	Not applicable.	Yes. (It designs service providers and domain administrators.)	Not applicable.	Yes, fine-grained is supported. (It supports the way to forward access control policies to other domains.)
RBAC-IC	Yes. (Expand the role to abstract roles and specific roles, and allow the specific role under different domains to have the same name.)	Yes. (Administrators are divided into platform administrators and domain administrators to handle specified businesses, respectively.)	Yes. (Different specific roles are only valid in the domain they belong to.)	Fine-grained is not supported. (The domain administrator can authorize other domain users to have specific roles, and will not authorize the role assignment right.)

6.2.1. Model Confidentiality Analysis

1. Unauthorized access;

$$sr \notin user\_authorization(u) \wedge s \in user\_session(u) \wedge sr \in session\_sr(s) \tag{27}$$

This formula describes that if user u does not authorize the specific role sr, but the user still activates session s, sr is used in session s. This indicates that the user has used an unauthorized specific role for unauthorized access.

Proof that RBAC-IC does not meet Formula (27).
Assume that user u has authorized sr, that is,

$$sr \in user\_authorization(u) \wedge s \in user\_session(u) \wedge sr \in session\_sr(s) \tag{28}$$

is established;
When the sr is not authorized to the user, if there is a case where s can be activated by the user and sr can be used in s, Formula (27) holds;
At the same time. Illustrate the following formula,

$$sr \notin user\_authorization(u) \wedge sr \in user\_authorization(u) \wedge s \in user\_session(u) \wedge sr \in session\_sr(s) \tag{29}$$

should be established;
However, due to $sr \in user\_authorization(u)$ and $sr \notin user\_authorization(u)$, conflicts exist;
Therefore, Formula (29) does not hold; that is, Formulae (27) and (28) cannot hold at the same time;
It can be proved that RBAC-IC does not meet Formula (27). □

2. Access with specific role without permission;

$$p \notin sr\_assignment(sr) \wedge sr \in user\_authorization(u) \wedge s \in user\_session(u) \wedge p \in session\_permission(s) \tag{30}$$

This formula describes that if user u authorizes the specific role sr, sr does not assign permission p, but the user still activates session s, and p is used in session s. This indicates that the user has used an unassigned permission for unauthorized access.

The proof process for RBAC-IC not meeting Formula (30) is similar to Formula (27).

3. Access with expired specific role;

$$current\_time \notin sr[valid\_time] \wedge s \in user\_session(u) \wedge sr \in session\_sr(s) \tag{31}$$

This formula describes that the specific role sr has expired, but user u can still use sr in the activated session s. This indicates that the user is using an expired specific role.

Proof that RBAC-IC does not meet Formula (31).
Assume that the current time is within the effective time range of the specific role sr; that is, $sr[valid\_time][min\_time] \leq current\_time \leq \mathrm{sr}[valid\_time][max\_time]$;
If the current time is not within the valid time range; that is, $sr[valid\_time][min\_time] > current\_time > \mathrm{sr}[valid\_time][max\_time]$;
If there is a case where s can be activated by the user and sr can be used in s, Formula (31) holds;
At the same time. Illustrate the following formula,

$$current\_time \notin sr[valid\_time] \wedge current\_time \in sr[valid\_time] \wedge s \in user\_session(u) \wedge sr \in session\_sr(s) \tag{32}$$

should be established;
However, $current\_time \notin sr[valid\_time]$ and $current\_time \in sr[valid\_time]$ conflicts exist, and RBAC-IC stipulates that specific roles can only be used within the validity period.
Therefore, Formulae (31) and (32) cannot hold at the same time.
It can be proved that RBAC-IC does not meet Formula (31). □

6.2.2. Model Constraint Security Analysis

1. Static mutex constraint;

$$static\_mutex(sr_1, sr_2) \wedge sr_1, sr_2 \in user\_authorization(u) \tag{33}$$

where $static\_mutex(sr_1, sr_2)$ indicates that specific roles sr_1 and sr_2 have static mutual exclusion constraints; that is, the two roles cannot be authorized to one user. This formula describes the sr_1 and sr_2 that authorize the user u with static mutex constraints at the same time. This indicates that the static mutual exclusion constraint rule of the specific role is violated.

Proof that RBAC-IC does not meet Formula (33).
Assume there is user u, and specific roles sr_1 and sr_2, $sr_1 \neq sr_2$, and there is a static mutual exclusion relationship between sr_1 and sr_2;
Assume that sr_1 is authorized to u, and sr_2 is not authorized to u; that is,

$$static\_mutex(sr_1, sr_2) \wedge sr_1 \in user\_authorization(u) \wedge sr_2 \notin user\_authorization(u) \tag{34}$$

is established;

If both sr_1 and sr_2 are authorized to u, Formula (31) is established;
According to the static mutual exclusion relationship between sr_1 and sr_2, users can only be authorized to a specific role in the specific role set with static mutual exclusion relationship;
Therefore, both sr_1 and sr_2 cannot be authorized to u;
It can be proved that RBAC-IC does not meet Formula (33). □

Figure 10 shows an example of a static mutex constraint.

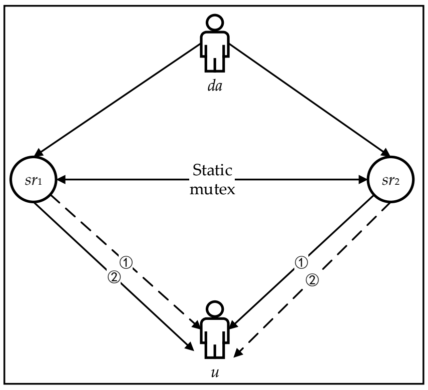

Figure 10. There is a static mutex constraint between sr_1 and sr_2. The domain administrator *da* can only authorize one of sr_1 and sr_2 to the user, that is, only one of ① and ② processes can be executed.

2. Cardinality constraint;

$$cardinality(sr, m) \wedge sr \in user\_authorization(u_1) \wedge sr \in user\_authorization(u_2) \wedge \ldots \wedge sr \in user\_authorization(u_n)\ (n > m) \quad (35)$$

where $cardinality(sr, m)$ means that the specific role sr can only be authorized to m users at most, and there is a cardinality constraint. This formula describes that the specific role sr is authorized to n users at the same time, and the number of authorizations exceeds the maximum number of cardinals m. This shows that the cardinal constraint rule of the specific role is violated.

Proof that RBAC-IC does not meet Formula (35).
Assume that the specific role sr exists, and sr has cardinality constraints $cardinality(sr, m)$;
In the current state, no user has authorized sr. At this time, if the domain administrator has authorized sr for no more than m users, the $cardinality(sr, m)$ constraint is satisfied; That is

$$cardinality(sr, m) \wedge sr \in user\_authorization(u_1) \wedge sr \in user\_authorization(u_2) \wedge \ldots \wedge sr \in user\_authorization(u_n)\ (n \leq m) \quad (36)$$

is established;
If the number of users with sr is greater than m, Formula (35) is valid;
According to the pigeonhole principle [31], if m users have $m + 1$ sr, it means that at least one user will have two identical sr, but the user cannot authorize this sr again if he has 1 sr;
Therefore, the number of users with sr cannot be greater than m;
According to $cardinality(sr, m)$, sr can only be authorized to m users at most, and no more than m users have sr.
Therefore, Formula (35) conflicts with cardinal constraint rules, and Formula (35) is not valid;
It can be proved that RBAC-IC does not meet Formula (35). □

3. Prerequisite constraint;

$$prerequisite(sr_1, sr_2) \wedge sr_2 \notin user\_authorization(u) \wedge sr_1 \in user\_authorization(u) \quad (37)$$

where $prerequisite(sr_1, sr_2)$ indicates that the user must have the specific role sr_2 before authorizing the specific role sr_1, and sr_1 has prerequisite constraints. This formula describes that the user u is authorized with sr_1, but not with sr_2. This indicates that the precondition constraint rule of the specific role is violated.

Proof that RBAC-IC does not meet Formula (37).
Assume there is user u, and specific roles sr_1 and sr_2, $sr_1 \neq sr_2$, and sr_2 is a prerequisite role of sr_1;
Follow these steps to authorize users,
step1: Authorize sr_2 to u;
step2: Authorize sr_1 to u;
If step 1 is never executed and step 2 is directly executed, Formula (37) is established;
According to the prerequisite of the specific role, the user must have authorized the specific role that has the prerequisite specific role.
Therefore, step2 cannot be executed without step1;
It can be proved that RBAC-IC does not meet Formula (37). □

Figure 11 shows an example of a prerequisite constraint.

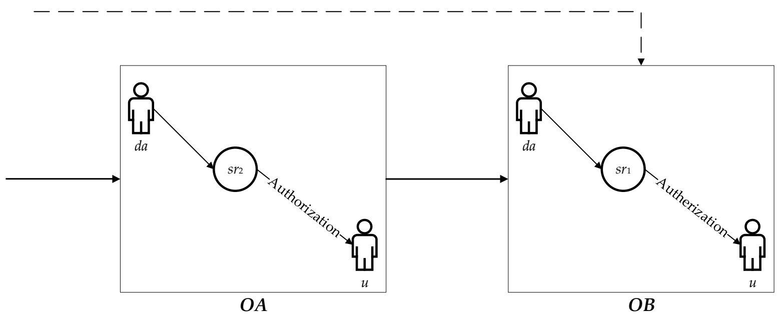

Figure 11. sr_2 is a prerequisite role of sr_1. The *OA* operation indicates that the domain administrator *da* authorizes sr_2 to u, and the *OB* operation indicates that the domain administrator *da* authorizes sr_1 to u. Before performing the *OB* operation, *OA* must be executed, and the *OA* operation cannot be skipped to execute *OB*.

4. Dynamic mutex constraint;

$$dynamic\_mutex(sr_1, sr_2) \wedge sr_1, sr_2 \in user\_authorization(u) \wedge s \in user\_session(u) \wedge sr_1, sr_2 \in session\_sr(s) \tag{38}$$

where $dynamic\_mutex(sr_1, sr_2)$ indicates that specific roles sr_1 and sr_2 have a dynamic mutual exclusion constraint; that is, two roles cannot be used by user in one session. This formula describes that user u is authorized sr_1 and sr_2 with dynamic mutual exclusion constraint at the same time, and session s is activated, but user can use sr_1 and sr_2 at the same time in s. This shows that the dynamic constraint rules of the specific role are violated.

Proof that RBAC-IC does not meet Formula (38).
Assume there is user u, and the specific roles sr_1 and sr_2, $sr_1 \neq sr_2$, sr_1 and sr_2 have a dynamic mutual exclusion relationship, and sr_1, $sr_2 \in user\_authorization(u)$, sr1 and sr2 cannot exist in the same session at the same time,
If both sr_1 and sr_2 exist in session s, Formula (38) holds;
According to the dynamic mutual exclusion relationship between sr_1 and sr_2, sr_1 and sr_2 cannot exist in a session.
Therefore, Formula (38) is not valid;
It can be proved that RBAC-IC does not meet Formula (38). □

Figure 12 shows an example of a dynamic mutex constraint.

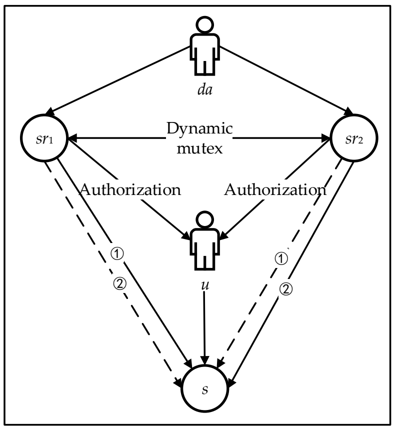

Figure 12. There is a dynamic mutual exclusion constraint between sr_1 and sr_2. The domain administrator *da* allows to authorizes both sr_1 and sr_2 roles to user *u*. However, after *u* activates session *s*, only one of sr_1 and sr_2 can exist in *s*; that is, only one of ① and ② processes can be executed.

6.2.3. Model Cross-Domain Security Analysis

For cross-domain authorization and cross-domain access in the model, security interoperability needs to be met. For authorization, the domain administrator cannot authorize the specific role of other domains to users. For access, users cannot hold the specific role or permission of other domains to access the objects of the requesting domain.

1. Cross-domain authorization;

$$(da[category] = domain\_administer \wedge da[domain] = domain_1 \wedge sr[domain] = domain_2) \wedge authorization(da, u, sr) \quad (39)$$

where *authorization*(*da*, *u*, *sr*) means that the domain administrator *da* authorizes the specific role *sr* for user *u*. This formula describes that *da* is the domain administrator of $domain_1$, *sr* is the specific role in $domain_2$, and *da* authorizes *sr* to user *u*. This indicates that the domain administrator authorizes the specific role of other domains to users, which violates the inter-domain isolation management.

Proof that RBAC-IC does not meet Formula (39).
Assume domain administrator *da* and user *u* exist in $domain_1$, and the specific role *sr* exists in $domain_2$. *da* authorizes *sr* to *u*, and Formula (39) is established;
However, RBAC-IC requires that the domain administrator can only authorize the specific roles in the local domain to users;
Formula (39) does not meet the above rules, and the *authorization*(*da*, *u*, *sr*) operation cannot be executed;
It can be proved that RBAC-IC does not meet Formula (39). □

Figure 13 shows an example of cross-domain authorization.

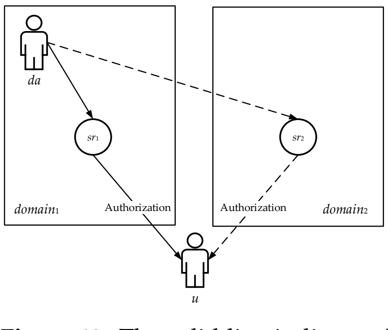

Figure 13. The solid line indicates that it is feasible for domain administrator *da* to authorize the specific role sr_1 of local domain to user *u*. The dotted line indicates that the *da* authorizes the sr_1 of other domains to the *u*, but it is not feasible.

2. Cross-domain access (Wrong specific roles);

$$(sr[domain] = domain_1 \wedge o[domain] = domain_2) \vee (sr[system] = system_1 \wedge o[system] = system_2) \wedge s \in user\_session(u) \wedge sr \in session\_sr(s) \wedge access(u, sr, o) \tag{40}$$

where $access(u, sr, o)$ means that user u uses the specific role sr to access object o. This formula describes that the specific role sr can be used in $domain_1$ or $system_1$. User u has activated session s, sr is allowed in s and u successfully uses sr to access object o, but o is stored in a system in $domain_2$ or $system_2$ in a domain. This indicates that the user has performed an illegal cross-domain operation.

When RBAC-IC uses a specific role to access an object, the specific role cannot be different from the domain or system of the target object. Figure 14 shows an example of cross-domain access, which illustrates that RBAC-IC does not satisfy Formula (40).

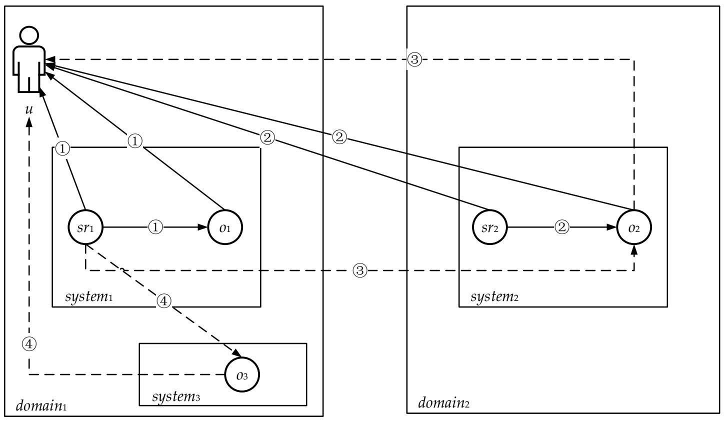

Figure 14. ① Indicates that user u accesses object o_1 using the specific role sr_1 of $system_1$ under $domain_1$, which is allowed. ② Indicates that u uses the specific role sr_2 of $system_2$ under $domain_2$, to access object o_2 across domains, which is also allowed. ③ Indicates that sr_1 is not allowed to access o_2 across domains. ④ Indicates that it is not allowed to use sr_1 to access object o_3 in $system_3$ across systems.

3. Cross-system access (Wrong permissions);

$$(p[domain] = domain_1 \wedge o[domain] = domain_2) \vee (p[system] = system_1 \wedge o[system] = system_2) \wedge s \in user\_session(u) \wedge sr \in session\_permission(s) \wedge access(u, p, o) \tag{41}$$

where $access(u, p, o)$ indicates that user u accesses object o with permission p. This formula describes that permission p can be used in $domain_1$ or $system_1$, user u has activated session s, p is allowed in s and u has successfully used p to access object o, but o is stored in a system in $domain_2$ or $system_2$ in a domain. This indicates that the user has performed an illegal cross-domain operation.

Explanation and examples of Formula (41) is similar to Formula (40).

6.2.4. Model Platform–Domain Isolation Security Analysis

$create\_ar(da, ar)$ indicates that domain administrator da creates abstract role ar. This formula describes the abstract role ar created by the domain administrator da. This violates the platform–domain isolation management. In addition, the security violation formula of the domain administrator performs other operations on platforms, and the platform administrator's operations in the domain are similar to Formula (42).

$$da[category] = domain\_administer \wedge create\_ar(da, ar) \tag{42}$$

Assume there is a domain administrator da in a domain under a platform. If the domain administrator creates an abstract role ar, Formula (41) holds. However, according to the platform–domain isolation management feature in RBAC-IC, domain administrators cannot perform platform level operations, such as creating abstract roles. Therefore, RBAC-IC does not satisfy Formula (38).

Figure 15 shows an example of platform–domain isolation management.

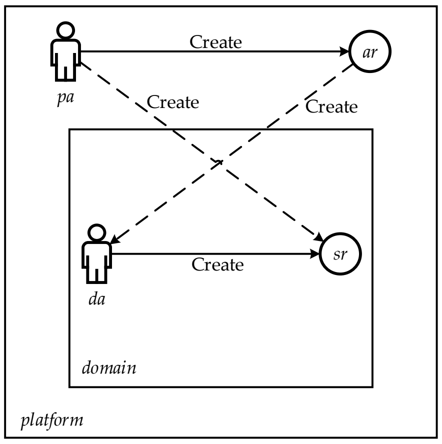

Figure 15. The platform administrator *pa* creates an abstract role and domain administrator *da* creates a specific role are allowed. However, platform administrators create specific roles, and domain administrators create abstract roles, which are not allowed.

7. Conclusions and Future Work

A role-based access control model for inter-system cross-domain in multi-domain environment (RBAC-IC) is proposed. This model is based on the traditional RBAC model, where roles are divided into abstract roles and specific roles play different functions; users are divided into three categories: platform administrators, domain administrators and ordinary users. This paper explains the concepts of the model through formal definitions, and it expounds upon different functions of abstract roles and specific roles. The execution process of RBAC-IC is described by pseudocode. RBAC-IC has four features that support role name repetition, platform–domain isolation management, inter-domain isolation management, and fine-grained cross-domain sharing. RBAC-IC carried out a typical case verification under the information service platform of the packaging can manufacturing group, and it conducted model security analysis by establishing security violation formulas. It is proved that RBAC-IC can run securely, and it is suitable to be used as the access control model of multi-domain information service platform to address the issue of inter-system cross-domain access control.

The security analysis of the model still needs further research, especially in the aspect of cascading security threats [32]; for example, the security events of system X in domain A may affect other systems in other domains. In the future work, we will carry out more research in this area.

Author Contributions: Conceptualization, Y.L. and Z.D.; methodology, Y.L. and L.L.; software, Y.L.; validation, Y.L. and L.L.; formal analysis, Y.L. and L.L.; writing—original draft preparation, Y.L., L.L. and Z.D.; writing—review and editing, Y.L. and Z.D.; supervision, Z.D. and Y.F.; funding acquisition, Y.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Shaanxi Natural Science Basic Research Project (Grant Number: 2021KW-07) and the Shaanxi International Science and Technology Cooperation Program Project (Grant Number: 2022QFY01-14).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors thank Timechainer (Beijing) Technology Co., Ltd. and its general manager Mao Ye for their support in software development. The authors thank Taishan Zhizhen Packaging Technology Co., Ltd. and Zhipeng Cai of North Automatic Control Technology Institute for their support in verification.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jiao, T. Mobile English Teaching Information Service Platform Based on Edge Computing. *Mob. Inf. Syst.* **2021**, *2021*, 2082282. [CrossRef]
2. Pynnönen, S.; Haltia, E.; Hujala, T. Digital forest information platform as service innovation: Finnish Metsaan.fi service use, users and utilisation. *For. Policy Econ.* **2021**, *125*, 102404. [CrossRef]
3. Qian, C.; Zhang, Y.; Liu, Y.; Wang, Z. A cloud service platform integrating additive and subtractive manufacturing with high resource efficiency. *J. Clean. Prod.* **2019**, *241*, 118379. [CrossRef]
4. Liu, B.; Chen, H.; Junmei, H. Design and Implementation of University Continuing Education Informatization Platform Based on SaaS Model. In Proceedings of the 2020 15th International Conference on Computer Science & Education (ICCSE), Delft, The Netherlands, 18–22 August 2020; pp. 253–256. [CrossRef]
5. Mahalle, A.; Yong, J.; Tao, X. Challenges and Mitigation for Application Deployment over SaaS Platform in Banking and Financial Services Industry. In Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Dalian, China, 5–7 May 2021; pp. 288–296. [CrossRef]
6. Celesti, A.; Ruggeri, A.; Fazio, M.; Galletta, A.; Villari, M.; Romano, A. Blockchain-Based Healthcare Workflow for Tele-Medical Laboratory in Federated Hospital IoT Clouds. *Sensors* **2020**, *20*, 2590. [CrossRef]
7. Wen, J.; Deng, B.; Peng, L.; Zhang, Y.; Zhang, B. Building of SaaS platform of hospital operational risk monitoring based on blockchain and smart contract. *J. Med. Inform.* **2019**, *40*, 18–22.
8. Ferraiolo, D.; Kuhn, D. Role-based access controls. In Proceedings of the 15th NIST-NCSC National Computer Security Conference, Baltimore, MD, USA, 13 October 1992; pp. 554–563.
9. Pan, N.; Sun, L.; He, L.-S.; Zhu, Z.-Q. An Approach for Hierarchical RBAC Reconfiguration with Minimal Perturbation. *IEEE Access* **2017**, *6*, 40389–40399. [CrossRef]
10. Ghafoorian, M.; Abbasinezhad-Mood, D.; Shakeri, H. A Thorough Trust and Reputation Based RBAC Model for Secure Data Storage in the Cloud. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *30*, 778–788. [CrossRef]
11. Thakare, A.; Lee, E.; Kumar, A.; Nikam, V.B.; Kim, Y.-G. PARBAC: Priority-Attribute-Based RBAC Model for Azure IoT Cloud. *IEEE Internet Things J.* **2020**, *7*, 2890–2900. [CrossRef]
12. Chao, L.; He, D.; Huang, X.; Choo, K.K.R.; Vasilakos, A. V B. SeIn: A blockchain-based secure mutual authentication with fine-grained access control system for industry 4.0. *J. Netw. Comput. Appl.* **2018**, *116*, 42–52. [CrossRef]
13. Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based access control models. *IEEE Comput.* **1996**, *29*, 38–47. [CrossRef]
14. Sandhu, R.; Bhamidipati, V.; Munawer, Q. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.* **1999**, *2*, 105–135. [CrossRef]
15. Ferraiolo, D.F.; Sandhu, R.; Gavrila, S.; Kuhn, D.R.; Chandramouli, R. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.* **2001**, *4*, 224–274. [CrossRef]
16. Balusamy, B.; Ramachandran, S.; Priya, N. Achieving fine-grained access control and mitigating role explosion by utilising ABE with RBAC. *Int. J. High Perform. Comput. Netw.* **2017**, *10*, 109–117. [CrossRef]
17. Uddin, M.; Islam, S.; Al-Nemrat, A. A Dynamic Access Control Model Using Authorising Workflow and Task-Role-Based Access Control. *IEEE Access* **2019**, *7*, 166676–166689. [CrossRef]
18. Zhang, C.X.; LI, J.F.; Liu, Y.; Zhao, W.D. Design and implementation of universal management system based on roles and scopes. *Comput. Eng.* **2008**, *34*, 47–49. [CrossRef]
19. Li, J.; Zhang, C. A three-dimensional role based user management model in web information systems. In *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 657–665. [CrossRef]
20. Li, J.; Liao, Z.; Zhang, C.; Shi, Y. A 4D-Role Based Access Control Model for Multitenancy Cloud Platform. *Math. Probl. Eng.* **2016**, *2016*, 2935638. [CrossRef]
21. Freudenthal, E.; Pesin, T.; Port, L.; Keenan, E.; Karamcheti, V. dRBAC: Distributed role-based access control for dynamic coalition environments. In Proceedings of the 22nd International Conference on Distributed Computing Systems, Vienna, Austria, 2–5 July 2002; pp. 554–563. [CrossRef]
22. Tang, B.; Li, Q.; Sandhu, R. A multi-tenant RBAC model for collaborative cloud services. In Proceedings of the 2013 Eleventh Annual Conference on Privacy, Security and Trust, Tarragona, Spain, 10–12 July 2013; pp. 229–238. [CrossRef]
23. Abdelfattah, D.; Hassan, H.A.; Omara, F.A. A novel role-mapping algorithm for enhancing highly collaborative access control system. *Distrib. Parallel Databases* **2022**, *40*, 521–558. [CrossRef]

24. Shafiq, B.; Joshi, J.; Bertino, E.; Ghafoor, A. Secure interoperation in a multidomain environment employing RBAC policies. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 1557–1577. [CrossRef]
25. Du, S.; Joshi, J.B.D. Supporting authorization query and inter-domain role mapping in presence of hybrid role hierarchy. In Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies, Lake Tahoe, CA, USA, 7–9 June 2006. [CrossRef]
26. Gouglidis, A.; Mavridis, I.; Hu, V.C. Security policy verification for multi-domains in cloud systems. *Int. J. Inf. Secur.* **2013**, *13*, 97–111. [CrossRef]
27. Uikey, C.; Bhilare, D.S. RBACA: Role-based access control architecture for multi-domain cloud environment. *Int. J. Bus. Infor-Mation Syst.* **2018**, *28*, 1–17. [CrossRef]
28. Qi, H.; Di, X.; Li, J. Formal definition and analysis of access control model based on role and attribute. *J. Inf. Secur. Appl.* **2018**, *43*, 53–60. [CrossRef]
29. Geethakumari, G.; Negi, A.; Sastry, V.N. A cross-domain role mapping and authorization framework for RBAC in grid systems. *Int. J. Comput. Sci. Appl.* **2009**, *6*, 1–12.
30. Denning, P.J. Fault tolerant operating systems. *ACM Comput. Surv. CSUR* **1976**, *8*, 359–389. [CrossRef]
31. Trybulec, W.A. Pigeon hole principle. *J. Formaliz. Math.* **1990**, *2*, 575–579.
32. Ebad, S.A. Security assessment of large-scale IT infrastructure. *Sci. J. King Faisal Univ. Basic Appl. Sci.* **2021**, *22*, 136–143. [CrossRef]