Towards Role-based Authorization for OSGi Service Environments

Gail-Joon Ahn  
Arizona State University  
gahn@asu.edu

Hongxin Hu  
Arizona State University  
hongxinh@asu.edu

Jing Jin  
UNC Charlotte  
jjin@uncc.edu

Abstract

OSGi framework enables diverse devices to conveniently establish a local area network environment such as homes, offices, and automobiles. Access control is one of the crucial parts which should be considered in such emerging environments. However, the current OSGi authorization mechanism is not rigorous enough to fulfill security requirements involved in dynamic and open OSGi environments. This paper provides a systematic way to adopt a role-based access control approach in OSGi environments. We demonstrate how our authorization framework can achieve important RBAC features and enhance existing primitive access control modules in OSGi service environments.

1. Introduction

The Open Services Gateway Initiative (OSGi) provides an open, common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a cooperative fashion. In addition, OSGi service platform is an extensible integration platform used to remotely and dynamically deploy, provision, maintain and manage applications and services with multiple devices in networked environments [1]. In OSGi service platform, the User Admin service provides user authorization functionality. However, the User Admin service's authorization architecture in OSGi is not sufficient enough to support the highly dynamic and open OSGi environments. Thus, an enhanced access control mechanism is needed for achieving interoperability, extensibility, and ease of administration & management. To overcome such issues, role-based access control (RBAC) can be applied in such network environments to simplify authorization management by associating users with roles, and roles with permissions [10]. Because the roles within an organization typically have overlapping permissions, RBAC supports a hierarchical structure of roles to provide permission inheritance, where a senior role can inherit all permissions assigned to junior roles. As a fundamental aspect of RBAC, constraints can also allow us to lay out higher-level access control policies [3].

In this paper, we propose a systematic way to support a well-known access control in OSGi environments. In our proposed approach, the OSGi authorization mechanism is configured and mapped to RBAC, demonstrating that OSGi authorization requirements can be fully fulfilled by RBAC. In addition, important RBAC features are studied to enhance existing access control modules in OSGi service environments.

In the remainder of this paper, we first briefly describe current OSGi authorization mechanism. In Section 3, we formalize OSGi authorization mechanism and demonstrate how OSGi authorization can be accommodated in RBAC. The implementation of our prototype is addressed in Section 4. Several related work are discussed in Section 5. Finally, Section 6 concludes this paper.

2. OSGi Authorization Mechanism

In OSGi service platform, the User Admin service is utilized to find out if the users attempting to access are authorized or not [9]. In the User Admin service authorization architecture, three components are defined:

- **User** - A user is a human being who can be identified by credentials such as a password and other identity attributes.
- **User Group** - A user group is an aggregation of users based on common properties. For example, all family members belong to a user group named Residents.
- **Action Group** - Every action that can be performed by a bundle is associated with an action group. For example, if a bundle could be used to control the alarm system, there should be an action group named AlarmSystemControl.

In OSGi authorization, the authorization decision is made based on the following two strategies:

- **ANY Strategy**: a user could be allowed to carry out an action if s/he belongs to Any member of the action group. For example, the AlarmSystemControl action group contains two user groups Administrators and Residents. Elmer, Pepe and Bugs belong to Administrators user group, and
Elmer, Pepe and Daffy belong to Residents user group as follows:

- **Administrators** = \{ Elmer, Pepe, Foghorn \}
- **Residents** = \{ Elmer, Pepe, Daffy \}
- **AlarmSystemControl** = \{ Administrators, Residents \}

This ANY strategy allows any of four members, Elmer, Pepe, Foghorn and Daffy, to activate the alarm system since all users are a member of user groups and those user groups belong to one action group.

**ALL Strategy**: a user is allowed to carry out an action if s/he belongs to All members of the action group. In the above-mentioned AlarmSystemControl example, only Elmer and Pepe would be authorized to activate the alarm system, since Daffy and Bugs are not members of both Administrators and Residents user groups.

The implementation of User Admin service in OSGi service platform supports a combination of both strategies by introducing two member sets, namely the basic member set for the ANY strategy and the required member set for ALL strategy. Basic membership defines a set of members that can get access and required membership reduces this set by requiring a user to be a required member of each action group.

To accommodate this, OSGi assigns a user to a user group then the user group is assigned to a specific action group. Table 1 and 2 show an authorization example to demonstrate the assignment relationships respectively.

The access decision is made based on the basic and required assignment relationships. For example, in Table 2 the action group WebcamAccess has two basic members, Residents and Buddies, and two required members, Adults and Administrators. Thus, all users belonging to at least one of the basic members, Residents and Buddies, and all required members, Adults and Administrators, are able to carry out the webcam access action. From Table 1, we notice that Elmer and Foghorn can meet this authorization requirement.

### 3. Construction of OSGi-compliant RBAC

We found out that the current OSGi authorization mechanism is not intuitive for authorization administration. Furthermore, it is not suitable for satisfying all security requirements for defining fine-grained access control policies in a highly dynamic and open OSGi environment. Our objective is to provide an efficient and effective authorization mechanism for OSGi enabled network environments. To achieve this, we adopt a role-based access control (RBAC) which is a powerful mechanism for reducing the management complexity, administration cost and potential configuration error within the organization [10]. Hence, it is inevitable to identify and derive RBAC components from OSGi authorization requirements.

We first attempt to formally define components in the current OSGi authorization mechanism and these formal definitions are used through the rest of this paper. There are three sets of entities: users (U), user groups (UG) and action groups (AG). The basic user-to-user group assignment (BUA) is a many-to-many relation between U and UG. There are two kinds of many-to-many relation between UG and AG. One is basic action group-to-user group assignment (BAA) that reflects the basic memberships for each action. Another is required action group-to-user group assignment (RAA) that reflects the required memberships for each action. The following definitions formally summarize each component:

- \( U \) is a set of users, \( U = \{ u_1, \ldots, u_n \} \).
- \( UG \) is a set of user groups, \( UG = \{ u_{g1}, \ldots, u_{gn} \} \).
- \( AG = AG_{Basic} \cup AG_{Required} \) is a set of basic and required action groups, \( AG = \{ ag_{1}, \ldots, ag_{m} \} \).
- \( BUA \subseteq U \times UG \) is a many-to-many basic user-to-user group assignment relation.
- \( BAA \subseteq AG_{Basic} \times UG \) is a many-to-many basic action group-to-user group assignment relation, \( BAA[ag_{i}, u_{gj}] = \{ \text{basic} \} \).
- \( RAA \subseteq AG_{Required} \times UG \) is a many-to-many required action group-to-user group assignment relation, \( RAA[ag_{i}, u_{gj}] = \{ \text{required} \} \).
- \( \text{users} : UG \rightarrow 2^U \) is a function mapping each user group \( u_{g} \) to a set of users, \( \text{users}(u_{gj}) = \{ u \in U | (u, u_{gj}) \in BUA \} \).
- \( \text{user groups} : U \rightarrow 2^{UG} \) is a function mapping each user \( u_{i} \) to a set of user groups, \( \text{user groups}(u_{i}) = \{ u_{g} \in UG | (u_{i}, u_{g}) \in BUA \} \).
- \( \text{ba_user groups} : AG_{Basic} \rightarrow 2^{UG} \) is a function mapping each basic action group \( ag_{i} \) to a set of user groups, \( \text{ba_user groups}(ag_{i}) = \{ u_{g} \in UG | (ag_{i}, u_{g}) \in BAA \} \).
3.1. Constructing basic RBAC components

The essence of RBAC is the notion of roles to abstract users and permissions. Permissions are grouped through roles, and users obtain permissions by being assigned to roles. Users, roles, permissions and the corresponding assignment relations are core components in RBAC.

We noticed that the User set (U) and the Action Group set (AG) essentially can be treated as users and permissions in the RBAC model, respectively. In addition, the User Group set (UG) in OSGi can be represented as roles in such that actions are abstracted through User Groups and those actions can be exercised by a member of the specific User Group. However, normally OSGi uses two types of assignments, namely the basic assignment (BAA) and the required assignment (RAA) that can allow us to achieve the abstraction of actions through User Groups. However, RBAC cannot accommodate this characteristic directly. So, we propose several properties to bridge this gap in the course of OSGi-compliant RBAC construction.

First, we construct the role (R) by introducing a concept of Private Membership. In particular, we treat the user groups in OSGi as the private members of each role in role construction, which can be further characterized as basic members and required members depending on the property of the OSGi BAA relation and RAA relation. The detailed role construction process is explained through an OSGi authorization example as shown in Figure 1. In this example, Table (a) reflects the assignment relation between U and UG, and Table (b) shows the assignment relation between AG and UG. In Table (b), there are two types of assignment relations, Basic and Required to reflect BAA and RAA relations respectively. For a particular action group, we identify that there exist three possible combinations on Basic and Required assignments as follows:

Case 1: Contains both basic assignments and required assignments as shown in the row of
\(a_{g_1}\), \(ba_{user\_groups}(a_{g_1}) = \{u_{g_1}, u_{g_2}\}\) and \(re_{user\_groups}(a_{g_1}) = \{u_{g_4}, u_{g_5}\}\).

**Case 2:** Contains only basic assignments as shown in the row of \(a_{g_3}\), \(ba_{user\_groups}(a_{g_3}) = \{u_{g_1}, u_{g_2}, u_{g_3}\}\).

**Case 3:** Contains only required assignments as shown in the row of \(a_{g_2}\), \(re_{user\_groups}(a_{g_2}) = \{u_{g_1}, u_{g_4}, u_{g_5}\}\).

As specified in the OSGi authorization strategies, a user must be assigned to ALL required user groups and ANY basic user groups before s/he can exercise a particular action. Also, while accommodating the identified basic and required assignment patterns, we define the following mapping rules on action group basis:

**Rule 1.** Introduce one role and one permission-to-role assignment for each OSGi basic action group-to-user group assignment in BAA. The role contains only one basic private member of the basic user group and ALL required members of the required user groups. The particular action group in BAA is constructed as one permission and the permission is assigned to the role.

**Rule 2.** In case of no basic assignment, introduce one role that contains ALL required members of the required user groups and one permission-to-role assignment by assigning the permission to the role.

**Rule 3.** In terms of user-to-role assignment, a user can be assigned to a role when the corresponding user in OSGi authorization is assigned to all the private members of that role.

Now we discuss how to use these three rules to construct the RBAC for each case we have identified. First, we map all users and action groups in OSGi authorization to users and permissions of RBAC directly. As identified in Case 1, the action group \(a_{g_1}\) has both basic member and required member. Following Rule 1, we construct roles such as two roles \(r_1\) and \(r_2\) for the action group \(a_{g_1}\) in our example. \(r_1\) contains a basic member \(u_{g_1}\) and two required members \(u_{g_4}\) and \(u_{g_5}\), \(r_2\) is constructed by a basic member \(u_{g_2}\) and all required members \(u_{g_4}\) and \(u_{g_5}\). Formally, \(members(r_1) = \{u_{g_1}, u_{g_4}, u_{g_5}\}\), and \(members(r_2) = \{u_{g_2}, u_{g_4}, u_{g_5}\}\). Using the same rule, the permission \(p_1\) corresponding to the action group \(a_{g_1}\) is assigned to roles \(r_1\) and \(r_2\). Following Rule 3, a user is assigned to a role only if the user has been assigned to all private members of this role. In the example (Figure 1 (a)), \(u_1\) is assigned to \(u_{g_1}, u_{g_4}\) and \(u_{g_5}\). Since \(u_{g_1}, u_{g_4}\) and \(u_{g_5}\) are private members of \(r_1\), \(u_1\) should be assigned to \(r_1\) in RBAC. For the same reason, \(u_5\) is assigned to \(r_2\). From Figure 2, \(u_1\) and \(u_5\) can hold the same action group \(a_{g_1}\), and \(u_1\) and \(u_5\) own the same permission \(p_1\) via \(r_1\) and \(r_5\), respectively. Hence, the same authorization requirements are fulfilled through the process of configuring RBAC.

In Case 2, the action group has only three basic members \(u_{g_1}, u_{g_2}\) and \(u_{g_3}\). Using Rule 1, three roles \(r_1, r_5\) and \(r_5\) are constructed by these three basic members of \(a_{g_3}\) as shown in Figure 2. Formally, \(members(r_1) = \{u_{g_1}\}\), \(members(r_5) = \{u_{g_2}\}\) and \(members(r_5) = \{u_{g_3}\}\). Corresponding assignment relations as a result of Rule 1 and 3 are shown as well.

In Case 3, the action group has only required members. The second row of the OSGi authorization example (Figure 1 (b)) shows an action group \(a_{g_2}\) has three required members \(u_{g_1}, u_{g_4}\) and \(u_{g_5}\). Using Rule 2, one role \(r_3\) is constructed that contains all three required members of \(a_{g_2}\). Formally, \(members(r_3) = \{u_{g_1}, u_{g_4}, u_{g_5}\}\). Figure 2 depicts the role construction and assignment relationships.

To realize the basic RBAC construction from OSGi authorization, a mapping algorithm is given in Figure 3, which can be performed to map OSGi authorization to ba-

---

**Figure 3. Mapping Algorithm from OSGi Authorization to RBAC Authorization.**
sic RBAC as we discussed above. Some elements and functions in these algorithms—such as USER, AG, user_groups(), ba_user_groups() and re_user_groups()—are adopted from our formal representation of OSGi authorization mechanism. Some other elements and functions are based on the formal RBAC model.

### 3.2. Supporting role hierarchies and constraints

We construct role hierarchy to define an inheritance relationship, reducing the cost of administration. For example, all managers in the same organization may have a certain set of core “management privileges”, even though they work in different departments. This commonality can be exploited through a role hierarchy that makes each department manager role to inherit a generic “management” role. Role hierarchy allows the policy designer to write generic access polices once by simplifying the complexity of access control policies.

In our mapping approach, roles are constructed based on basic members and required members of action groups of OSGi authorization mechanism. Considering the private member sets of every role, we discover inclusion relations between them. In Figure 2(a), \( r_1 \) is constructed by a basic private member \( u_{g1} \) and two required private members \( u_{g4} \) and \( u_{g5} \), while \( r_7 \) is constructed by a basic private member \( u_{g1} \) and a required private members \( u_{g4} \). Formally, the following condition is true.

\[
\text{ba}_\text{member}(r_7) = \text{ba}_\text{member}(r_1) \land \text{re}_\text{members}(r_7) \subseteq \text{re}_\text{members}(r_1)
\]

Since the private members of \( r_1 \) include all private members of \( r_7 \), a user who is assigned to \( r_1 \) should be assigned to \( r_7 \) as well according to Rule 3. Thus a user assigned to \( r_1 \) should have all permissions of \( r_7 \). In other words, \( r_1 \) should possess all permissions of \( r_7 \) and \( r_1 \) is more powerful than \( r_7 \). Therefore, a role hierarchy relation can be built between \( r_1 \) and \( r_7 \). As a senior role, \( r_1 \) inherits all permissions of \( r_7 \). Commonly, if any of the following two conditions is true, \( r_1 \) is senior to \( r_7 \). With this methodology, two role hierarchies can be identified and shown in Figure 2(c).

1. \((\text{ba}_\text{member}(r_j) \neq \phi) \land (\text{ba}_\text{member}(r_i) \neq \phi) \land (\text{re}_\text{members}(r_j) = \text{ba}_\text{member}(r_i)) \land (\text{re}_\text{members}(r_j) \subseteq \text{re}_\text{members}(r_i))\)
2. \((\text{ba}_\text{member}(r_j) = \phi) \land (\text{re}_\text{member}(r_i) = \text{ba}_\text{member}(r_i)) \land (\text{re}_\text{members}(r_j) = \phi) \land (\text{re}_\text{members}(r_j) \subseteq \text{re}_\text{members}(r_i))\)

**RBAC** is a policy oriented approach and allows to specify well-known security principles, such as separation of duty and least privilege. These security principles can be defined as constraints in RBAC. Also we have proved that OSGi authorization requirements can be satisfied by RBAC in the previous discussion. We briefly elaborate how RBAC constraints can help enhance OSGi authorization in this section.

**Separation of duty (SOD).** SOD is a well-known principle for preventing fraud by identifying conflicting roles. In home network environments, some roles cannot be assigned to the same user, such as Residents and Buddies, Adults and Children. These roles can be defined as conflicting roles and the corresponding SOD constraints help avoid having undesired assignments.

**Prerequisite constraints.** This constraint is based on the concept of prerequisite roles. For example, a user can be assigned to the Administrators role only if the user is already a member of the Residents role. It ensures that only users who are already assigned to the Residents role can be assigned to the Administrators role.

### 4. Implementation Details

In order to prove the feasibility of our approach, we design a prototype system. A security-enhanced OSGi authorization architecture is depicted in Figure 4. This authorization architecture is composed with two distinct parts, policy management and policy enforcement.

#### 4.1. Policy management

OSGi authorization mechanism has been defined in the standard of OSGi service platform specification [9]. However, there is lack of available tools for managing OSGi authorization. In our implementation, a web-based OSGi authorization management tool is designed to support our policy management framework. The tool is composed of two major GUIs: (1) OSGi authorization management GUI communicates with OSGi authorization engine to manage standard OSGi authorization policy and (2) RBAC authorization management GUI is used to manage RBAC authorization through the RBAC authorization engine. Transformation handler is responsible for mapping OSGi authorization to RBAC components, building role hierarchies and restructuring OSGi authorization by performing corresponding algorithms.

**XACML** [8] is a standard and general purpose access control policy language defined in XML, which is flexible enough to accommodate most access control policy needs. Since core and hierarchical RBAC implementation can be specified by using XACML [7], the XACML policy generator generates XACML policies from RBAC authorization constraints.
specification. Three types of XACML policies are generated: Role Policy Set, Permission Policy Set and Role Assignment Policy Set. All generated policies are stored in an authorization policy DB.

4.2. Policy enforcement

A typical authorization system includes a policy decision point (PDP) and a policy enforcement point (PEP). Existing solutions in authorization management can be divided into two kinds of architectures: pull mode and push mode. In push approach, each subject presents the required information to the PDP and the decision is sent to PEP, while the PEP collects the related information of subjects and queries the PDP for policy decision in pull approach.

Our authorization architecture currently utilizes the pull mode. By integrating the Sun’s XACML library [2] into OSGi service platform, the Authorization Service as the PDP module interprets XACML policies and makes access decisions. The User Admin Service as the PEP module queries the Authorization Service and enforces the relevant operations.

5. Related Work

Several related work investigated the access control mechanism for home network environments where the OSGi service platform is operated. Cho et al. [4, 5] proposed an authorization policy management framework based on RBAC for OSGi service platform. Through the comparison of several access control models, they claimed that RBAC model is more flexible than DAC model for home network environments operated by the OSGi service platform. However, they did not consider the established authorization mechanism in current OSGi standard. Also, they omitted to address how important RBAC features, such as role hierarchy and constraints, can be effectively utilized in OSGi service platform. Lim et al. [6] presented a mechanism to bundle authentication and authorization services for the OSGi service platform. Their approach uses XACML to specify RBAC polices for the authorization of service bundles. However, they also ignored existing OSGi authorization mechanism and attempted to employ RBAC in the OSGi platform directly. To the best of our knowledge, our approach illustrated in this paper is the only RBAC authorization solution that is compatible to the existing OSGi authorization standard.

6. Concluding Remarks

In order to prove RBAC can fulfill OSGi authorization requirements and to leverage important RBAC features in the OSGi environment,

We have introduced a systematic mechanism to construct RBAC-compliant OSGi authorization modules. Our approach fulfilled OSGi authorization requirements while leveraging important RBAC features in the OSGi environment. A proof-of-concept prototype has been implemented to demonstrated the feasibility of our approach as well. Currently, we seek a way to validate whether changes in the RBAC-based environment can also be reflected in OSGi authorization environment. In addition, we plan to study
the specification of complicated access control policies, and verification and testing of access control policy specification in OSGi environments.

7. Acknowledgments

This work was supported, in part, by funds provided by National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

References