An XML framework for multi-level access control in the enterprise domain

Ioannis Priggouris[1], Stathes Hadjiefthymiades[1], Lazaros Merakos[1]

[1] University of Athens, Department of Informatics & Telecommunications, Panepistimioupolis, Ilissia, 15784, Athens, Greece

Abstract. Modeling security information has always been a fundamental part of every security system. A robust and flexible model is needed in order to guarantee both the easy management of security information and the efficient implementation of security mechanisms. In this paper, we present an XML-based framework, which can be used for controlling access to computer systems. The framework is mainly targeted to enterprise systems and aims to provide a fine-grained access control infrastructure for securing access to hosted services. The proposed framework supports both role-based and user-based access control on different levels. Although, the discussion focuses mainly on the data model, access control schemes and guidelines for implementing fitting security architectures are also provided.

1 Introduction

Secure service access comprises an area of extensive research and interest in the recent years. Different mechanisms and techniques have been adopted with the purpose of securing access to computer systems. However, apart from the implementation of the security mechanism a crucial issue in designing a robust security framework is the structure of the security meta-information, which is consulted in order to verify eligibility of a user for entering the system.

In this paper we present a framework, which can be used for implementing authentication and access control mechanisms over heterogeneous IT infrastructures. The framework defines the data structures needed for storing security information, as well as the actual process for implementing authorization and controlling access to specific resources. The data model specified using XML [7], which makes the architecture portable over different information repositories (xml files, RDBMSs, Directory Services, etc.). Our architecture is targeted to enterprise systems hosting multiple services. Its design is focused on providing a flexible scheme, which could sufficiently support such multi-service environments.

The rest of the paper is structured as follows. In section 2 a brief overview of related work and limitations of existing access control schemes is presented, followed by the description of our proposed model in section 3. The architectural aspects of the security infrastructure follows in section 4 and the paper concludes with a summary of the innovation achieved and a discussion on its potential application domain.

Priggouris I., Hadjiefthymiades S. and Merakos L. (2004).
An XML framework for multi-level access control in the enterprise domain.
In *Proceedings of the 2nd International Workshop on Security in Information Systems*, pages 227-236
DOI: 10.5220/0002675302270236
Copyright © SciTePress

2 Technology overview and related work

The simplest form of access control is the client authentication mechanism, which, however in its primitive form provides a flat security model. Nevertheless it can be augmented, with support for roles, in order to provide a multi-level security model, where access to individual resources is controlled separately. A role is an internal identity of the system, which defines the resources that a specific user is allowed to access. Role-based security is an elegant way to provide user authorisation and user access checks for an application. A user belonging to a particular role can access code, software and resources for which permissions are granted to the role. Incorporating roles makes security management much more flexible, while the security framework is rendered capable of supporting different security levels.

Role Based mechanisms for securing access to resources attracted significant research interest after 1990, when the concept of Role-Based Access Control (RBAC) emerged rapidly as a proven technology for managing and enforcing security in large-scale systems. A significant number of research papers on RBAC models and experimental implementations has been published in the recent past [1], [2], [3], [4], [5], [6]. A certain shortcoming of all these models is that they define RBAC mechanisms based on the assumption that roles have global scope. This assumption makes them inadequate for large enterprise environments, hosting multiple services, which are administered from different vendors. In such environments, using global roles is not advisable as their management may prove significant problem for the potential administrator, especially if the number of hosted services increases substantially. Evidently, a more flexible approach for controlling access to the resources hosted by such systems is needed.

3 Security Model

The model we propose is much more fine-grained than those available today. Each service defines specific roles, which are authoritative only within its context, having no impact on other services. Moreover, as discussed below, our architecture achieves all the above without restricting the potential namespace of the roles or the services. These characteristics are ideal for service provisioning platforms or other systems hosting varied functional entities, as it reduces drastically the administrative overhead needed for managing security roles. Moreover the model allows distributed management schemes to be adopted both for roles definition and for security enforcement. In such schemes, separate administrative entities can be responsible for specifying roles within a single service and assigning users to them, without caring if these roles have already been specified inside other services also.

Before delving further into the design aspects of the framework we will try to formalise it using propositional calculus. Our aim is to provide the basis for the design work that follows as well as a notation reference for future research in the same area. Similar formal approaches have been introduced in the past for equivalent architectures, such as the OASIS role-based access control framework [12]. Definitions of

basic concepts, like *services*, *methods*, *roles* and *users,* which will help the reader understand better the security architecture are also presented.

The model is based upon 6 basic sets:

- *U*: set of users
- *S*: set of services
- *M*: set of methods
- *G*: set of method signatures
- *R*: set of roles
- *N*: set of role names

A simple user *u* is an element of *U* ($u \in U$) and is defined as an entity, interacting with the protected computer system. The user usually is a human; however client programs or other computer systems can also be considered as users.

A service *s* is an element of *S* ($s \in S$), and corresponds to a software component running on a computer system. Borrowing the Object oriented terminology the service is the equivalent of an object and consists of several methods, which are the actual resources that need to be protected; since no other access in allowed to the service entity. Each method has a signature $g \in G$, which consists of its name and the list of invocation parameters. We won't delve further into the definition of the method signature, as it is not of prime importance to our model. A significant constraint of the model, we have presented so far, is that method signatures, although unique within the scope of each service, are not unique within the computer system. In order to surpass this constraint we use the pair $(s,g) \in S \times G$ introducing the concept of method *m* as an element of M ($m \in M$), which apparently bears global uniqueness because $M \subseteq S \times G$. Moreover, we denote as M_s the subset of methods belonging to the same service *s*. Evidently $M_s = \{m_1, m_2, ...m_k\}$ where $m_i \in M$ for $1 \le i \le k$ and $M_s \subseteq \{s\} \times G$ for each $s \in S$.

A role name *n* is an element of *N* ($n \in N$) and defines a logical label, which is used within a computer system for diversifying access to the hosted services. A role name is unique within the scope of its defining service. However, it can be re-used in the context of another service. In order to avoid confusion between the two definitions our model uses the pair $(s,n) \in S \times N$ in order to define a globally unique role $r \in R$.

In order to achieve the objective of protecting critical resources each role *r* is associated both with methods and users. Association with methods is used in order to determine the resources to be protected and will be hereafter referred to as *role declaration*. Association with users, on the other hand, defines the access rights to the method and will be referred as *role assignment*. In a more formal notation, a *role declaration* corresponds to a pair of $(r,m) \in R \times M$, while a *role assignment* to another pair $(r,u) \in R \times U$. In order for the user *u* to have access to a certain service method *m* both a *role declaration* and a *role assignment* for the same role *r* must have been defined within the model. Another association that can be defined in our model is that between users and services. This association, which will be referred to as *service eligibility,* is expressed in the form of pairs of $(s,u) \in S \times U$ and indicate that a user *u* is eligible to access the service *s*.

4 Architecture- Framework design

In this section we provide the foundations of our architectural approach and issues considered during the design phase. As already mentioned the security framework, supports the following 3 basic security operations:

- Authentication,
- Role-Based Access Control (RBAC)
- User-Based Access Control (UBAC)

Authentication is not actually covered by our model, but it is used in order to determine the identity of a user. The authentication mechanism undertakes the task of establishing a security context, which will carry all the privileges assigned to the specific user (e.g., roles). Of course these roles are specified inside the framework and should somehow be mapped to the specific application domain (e.g., the services). However this is an implementation specific issue, which should be considered when implementing the discussed framework. The simplest way to achieve this mapping is by hard-coding them inside the applications. Enterprise software offers alternative much more flexible ways, by using deployment descriptors ([9], [10]).

User-based access control is supported in two different levels:

- A low-level access control, which enables controlled access to the whole infrastructure.
- A high-level access control, providing a more fine-grained mechanism, which allows controlling access to a specific set of resources (i.e., a single service).

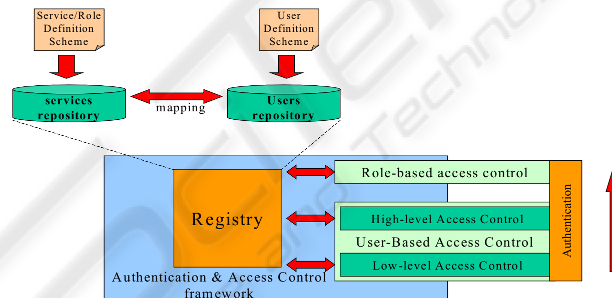

Fig. 1. Overall architecture

Role-based and user-based access control work independently of each other but they both rely on successful user authentication. Depending on the pursued functionality, the framework can be configured to enforce role-based or user-based access control only. The mechanisms could also stack in order to provide an integrated multi-level access control infrastructure. The proposed stack order is defined by the arrow in figure 1; low-level user-based access control is the most coarse security mechanism so it is the first to be invoked while role-based the most refined one and therefore it is placed last.

Delving inside the heart of the security architecture we find the Registry module. The Registry holds all information pertaining to potential users of the system, running

services and their roles. Moreover, the mapping between users and roles is harbored in it. The Registry is updated every time a new service (i.e., bundle of resources) is installed on the protected system. It is also updated each time a new user is defined as well as whenever the association between users and services is redefined. In subsequent sections the internal structure of the Registry will be presented in detail along with the key aspects that differentiate its design and allow it to fit in dynamic multi-service environments.

4.1 Registry

The Registry accommodates two repositories: one for services and roles and another for the users. Each Repository contains a set of entries of the same type. In order to populate the two repositories, specific schemes defining the structure of each stored element were designed. Specifically we defined:

- The User definition scheme
- The Service/Role definition scheme

Other information contained in the Registry includes the mapping between services, roles and users.

User definition scheme. This scheme specifies the way a user entry is stored. User entries act as a container for user-specific data. The defined scheme is fairly simple and can be seen in figure 2. Although the specification comes in the form of an XML schema [8], the presented framework does not consider any particular implementation. Thus, possible implementations may include xml files, RDBMS tables and LDAP objects.

Each user entry is identified by the unique *id* attribute and also has a unique *username* value. The framework uses the *id* attribute for internally discriminating between users, while the *username* is an easily memorized alias of the *id*. The scheme also defines an optional element for storing certificates, which can be used for supporting certificate-based client authentication. The rest of the fields (*name*, *surname*, *other-info, addresses* etc.) are rather trivial and are mainly used for storing supplementary information for each user.

Service/Role definition scheme. The Service definition scheme specifies the structure of the service entry, which provides a convenient storage scheme for service-specific data. The scheme can store various information elements pertaining to the service, as seen in figure 3. The existing information elements were adopted in order to apply the security framework in a service provisioning platform, where services were exposed through a web interface. However, the exact definition of the service scheme can vary according to the application domain as other applications may require additional data to be stored or render some of the existing elements obsolete.

The specification of the *roles* element is presented in figure 4. Individual roles are identified by an *id* attribute. The *id* corresponds to the role name (n), as defined in the formal model, whose uniqueness within the scope of the same service is enforced by

the proposed service specification. Embedding each role inside the service entry allows for the automatic pairing between service and role *ids* (i.e, the (*s,n*) pairs identifying the globally unique role *r*). A status attribute is supported for each role, allowing its enablement or disablement on-demand. The *role* is also the entity, which contains the actual association with the users (i.e. the *role assignment* that was defined in the formal model). In order to avoid duplicate *member* entries for each role, the corresponding element is marked as unique. The values of the *member* elements correspond to the *ids* of the users as the latter are defined inside the registry.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="addresses">
        <xs:complexType>
            <xs:all maxOccurs="unbounded">
                <xs:element name="address" type="xs:string" minOccurs="0"/>
            </xs:all>
        </xs:complexType>
    </xs:element>
    <xs:element name="user">
        <xs:complexType>
            <xs:all>
                <xs:element name="username" type="xs:string"/>
                <xs:element name="password" type="xs:string"/>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="surname" type="xs:string"/>
                <xs:element name="certificate" type="xs:string" minOccurs="0"/>
                <xs:element name="otherinfo" type="xs:string" minOccurs="0"/>
                <xs:element ref="addresses" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="id" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="users">
        <xs:complexType>
            <xs:all maxOccurs="unbounded">
                <xs:element ref="user" minOccurs="0"/>
            </xs:all>
        </xs:complexType>
        <xs:key name="idUniqueness">
            <xs:selector xpath="user"/>
            <xs:field xpath="@id"/>
        </xs:key>
        <xs:key name="usernameUniqueness">
            <xs:selector xpath="user"/>
            <xs:field xpath="username"/>
        </xs:key>
    </xs:element>
</xs:schema>
```

Fig. 2. User entry specification

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:import schemaLocation="accessControl.xsd"/>
    <xs:import schemaLocation="roles.xsd"/>
    <xs:simpleType name="sStatus">
        <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="STARTED"/>
            <xs:enumeration value="STOPPED"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="service">
        <xs:complexType>
            <xs:all>
                <xs:element name="description" type="xs:string" minOccurs="0"/>
                <xs:element name="url" type="xs:string"/>
                <xs:element name="referenceAddress" type="xs:string"/>
                <xs:element name="deploymentDate" type="xs:dateTime"/>
                <xs:element name="creator" type="xs:string"/>
                <xs:element name="otherinfo" type="xs:string" minOccurs="0"/>
                <xs:element ref="roles" minOccurs="0"/>
                <xs:element ref="accessControl" minOccurs="0"/>
            </xs:all>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="status" type="sStatus" use="optional" default="STOPPED"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="services">
        <xs:complexType>
            <xs:all maxOccurs="unbounded">
                <xs:element ref="service" minOccurs="0"/>
                <xs:element ref="accessControl" minOccurs="0"/>
            </xs:all>
        </xs:complexType>
        <xs:key name="serviceIDUnique">
            <xs:selector xpath="service"/>
            <xs:field xpath="@id"/>
        </xs:key>
    </xs:element>
</xs:schema>
```

Fig. 3. Service entry specification

The service specification contains also the appropriate information needed by the framework's access control mechanisms. Linking to these mechanisms is achieved through the defined *accessControl* element. The aforementioned element appears in two different levels within the service specification (see figure 3); one at the *services* level, which intends to cover low-level access to all possible resources (i.e., to the whole protected system) and a second one at the *service* level. The latter realises the *service eligibility* association, defined in our model, thus implementing the high-level access control that was defined in the beginning of the section.

The defined schema for the *accessControl* element is presented in figure 5. In order to support a flexible definition framework, the schema has the option of choosing between specifying either a list of users eligible to access the controlled resource or a list of non-eligible users. The appropriate information is stored under the *allowed* or *notAllowed* elements respectively, in the form of sets of user *ids*. The proposed scheme validates that the same *user* does not appear sin two different areas inside the same *accessControl* element, thus avoiding erroneous situations, where two conflicting restrictions apply to a single user.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="sBoolean">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="ENABLED"/>
      <xs:enumeration value="DISABLED"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="member" type="xs:string"/>
  <xs:element name="members">
    <xs:complexType>
      <xs:all maxOccurs="unbounded">
        <xs:element ref="member" minOccurs="0"/>
      </xs:all>
    </xs:complexType>
    <xs:unique name="NoDuplicateUsersPerRole">
      <xs:selector xpath="member"/>
      <xs:field xpath="."/>
    </xs:unique>
  </xs:element>
  <xs:element name="role">
    <xs:complexType>
      <xs:all>
        <xs:element name="description" type="xs:string" minOccurs="0"/>
        <xs:element name="otherinfo" type="xs:string" minOccurs="0"/>
        <xs:element ref="members" minOccurs="0"/>
      </xs:all>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="status" type="sBoolean" use="optional" default="ENABLED"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="roles">
    <xs:complexType>
      <xs:all maxOccurs="unbounded">
        <xs:element ref="role" minOccurs="0"/>
      </xs:all>
    </xs:complexType>
    <xs:unique name="uniqueRolesPerService">
      <xs:selector xpath="role"/>
      <xs:field xpath="@id"/>
    </xs:unique>
  </xs:element>
</xs:schema>
```

Fig. 4. Roles specification

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="sBoolean">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="ENABLED"/>
      <xs:enumeration value="DISABLED"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="notAllowed">
    <xs:complexType>
      <xs:all maxOccurs="unbounded">
        <xs:element name="user" type="xs:string" minOccurs="0"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="allowed">
    <xs:complexType>
      <xs:all maxOccurs="unbounded">
        <xs:element name="user" type="xs:string" minOccurs="0"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="accessControl">
    <xs:complexType>
      <xs:choice>
        <xs:element ref="allowed" minOccurs="0"/>
        <xs:element ref="notAllowed" minOccurs="0"/>
      </xs:choice>
      <xs:attribute name="status" type="sBoolean" use="optional" default="ENABLED"/>
    </xs:complexType>
    <xs:unique name="uniqueUserAllowed">
      <xs:selector xpath="allowed/user"/>
      <xs:field xpath="."/>
    </xs:unique>
    <xs:unique name="uniqueUserNotAllowed">
      <xs:selector xpath="notAllowed/user"/>
      <xs:field xpath="."/>
    </xs:unique>
  </xs:element>
</xs:schema>
```

Fig. 5. Access Control specification

An obvious omission, from our formal model is that no information concerning service methods is defined within the introduced XML specifications. Apparently, no *role declarations,* as defined in the formal model, exist but roles are directly associated with services instead of methods. Role declarations were deliberately not included in our proposed schemes as there are already related XML specifications, which are widely used today. The most noteworthy of these schemes is the EJB 2.x declarative security specification [10], [11] an example of which is cited in figure 6.

```xml
<assembly-descriptor>
  <method-permission>
    <role-name>Administrator</role-name>
    <method>
      <ejb-name>PositioningService</ejb-name>
      <method-name>getLocation</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
      </method-params>
    </method>
  </method-permission>
</assembly-descriptor>
```

Fig. 6. Declarative security definition in EJB 2.0 (excerpt from the EJB deployment descriptor)

4.2 Security mechanisms

A fundamental part of the security framework is the *security context,* which is created after a successful *id* is detected. The security context is an internal memory object indexed by the unique user *id* which holds all security information related to the spe-

cific user. An example of its structure (i.e., supported fields) is presented in figure 7. Following its creation, the security context is updated with the appropriate security information for the designated user.

Id	u1235678
System access	OK
Accessible Services	Service1
	Service4
	…
roles:	service1.role1
	service2.role1
	service2.role4
Valid until	12/4/2003 11:52

Fig. 7. Security Context definition

Each entry of the security context is filled with the appropriate information by the corresponding security mechanism. Low-level access control sets the *system access* field, while high-level access control updates the *accessible services* field with all services available to the user. Finally the role-based authorization process retrieves, from the registry, all the available roles for a specific user and inserts them in the *roles* field. The role of the *security context* is to provide some kind of caching mechanism for the information pertaining to the authenticated user in order to speed the authorization and access control process. It can be eliminated without any impact to the pursued functionality, but then each security mechanism will need to consult the Registry for every request submitted to the protected system, even if this request is the same with a previous one. The whole access control process is depicted in figure 8.

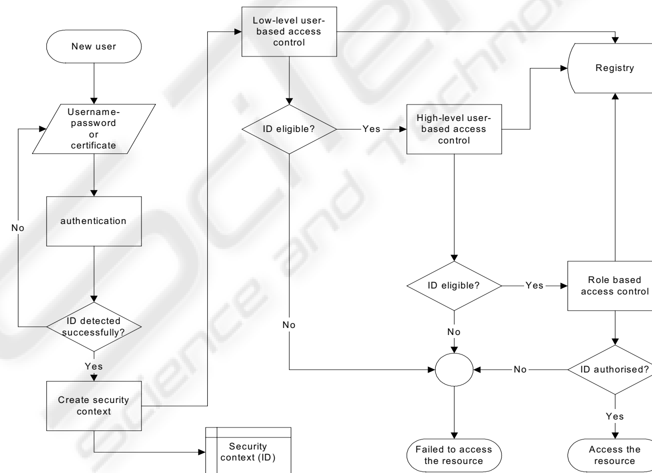

Fig. 8. Access control process

User-Based Access Control. It includes the low level and the high-level user-based access control mechanisms.

Low-level access control. The low-level access control is the first security mechanism, which can be applied in order to restrict/allow access to the whole protected system. The mechanism is automatically enabled if the accessControl element of the services portion of the Registry is present and set to ENABLED.

Low-level access control, searches all the entries under the aforementioned element for a *member* value that matches the *id* of the user who accesses the system. Depending on whether the *id* is a member of the *allowed* or *notAllowed* element the user can be granted or refused access to the rest of the resources. The mechanism takes also provision for updating the *system access* field of the security context with a Boolean *YES* or NO value. When a stack access control architecture like the one depicted in figure 1 is adopted, further invocation of subsequent access control mechanisms rely on the result produced by this security mechanism.

High-level access control. High-level access control is the second mechanism, which can be enforced. It performs the same operations with the low-level control, which was discussed in the previous section but on the service level this time. Depending on the implementation approach, high-level access control could process the whole Registry (i.e., all service entries) once and update the security context accordingly or perform this check on a per request basis each time access to a new service is requested. Before invoking a certain bundle of resources (e.g. a service), the mechanism checks whether the user is eligible to access the specific service and authorizes his further admission inside the service. Hereafter, the last mechanism (RBAC), undertakes the task of handling the user request.

Role-based Access Control. Supporting different roles per service is the key issue that differentiates the proposed framework from other security infrastructures. The RBAC mechanism performs a two phases process in order to determine if a user is eligible to access a resource inside the multi-service system.

In the first phase the roles of the authenticated user are retrieved and stored inside the *security context*. All *services* inside the Registry are sequentially processed, and if the particular *user* owns a specific role, the corresponding role name is appended in the list of *roles* of the *security context*. A role object consists of the role name, which is specified by the service administrator/creator, prefixed by the service name, thus forming the role r as defined in our model. The latter is unique inside the Registry, thus, guaranteeing also the global uniqueness of the role.

The second phase involves the actual access control process. At first, the required roles for accessing the resource/method are determined. This determination could vary depending upon the used *role declaration* scheme. For example, in J2EE environments required the roles could be retrieved, by searching inside the deployment descriptor (see figure 6). Subsequently, the required roles are checked against those present in the *security context*, which were retrieved during the first phase. If a match is found the user is authorized to access the resource. The second phase takes place every time a certain resource is accessed, while the first one only once when the RBAC mechanism is firstly invoked.

5 Conclusions

In this paper, we presented a security framework for controlling access to the critical resources of a computer system. We focused mainly on the definition of the appropriate data structures, which will accommodate the information needed for performing the required security checks. A configurable 3-layer resource access control mechanism, which allows implementation of security mechanism on two levels was also introduced. On the first level a coarse user-based access control is performed on the system's level, while on the second level a fine-grained role-based access control is performed on the service level. The most significant achievement of the framework is that it allows the definition of role names inside a certain service, without influencing other services running on the same computer system; yet each role maintains its uniqueness throughout the whole system, thus allowing the adoption of distributed (i.e., on a service level) role management schemes. The aforementioned characteristic is extremely important in enterprise systems and multi-service environments, as it can significantly reduce the administrative overhead needed for controlling access to their resources.

References

1. D. Ferraiolo, D. R. Kuhn: "Role based access control".,In Proceedings of the 15th Annual Conference on National Computer Security. National Institute of Standards and Technology, Gaithersburg, MD, 554–563,1992.
2. L. Guiri, "A new model for role-based access control", In Proceedings of the 11th Annual Conference on Computer Security Applications (New Orleans, LA, Dec. 1995).
3. L. Guiri, P. Iglio, "A formal model for role-based access control with constraints", In proceedings of 9th IEEE Workshop on Computer Security Foundations, Ireland, 1996.
4. S. Osborn, R. S. Sahdhu, Q. Mutanawer, "Configuring role-based access control to enforce mandatory and discretionary access control policies". ACM Trans. On Information System Security 3, 2 (May 2000).
5. I. Mohammed, D. M. Dilts, "Design for dynamic user-role-based security", Computer Security 13, 8, 661–671, 1994.
6. J. Park, R. Sandhu, G. Ahn, "Role-Based Access Control on the Web", ACM Transactions on Information and Systems Security (TISSEC), Volume 4, Number 1, February 2001.
7. M. Birbeck et al, "Professional XML", Wrox Press Inc, 1st edition, 2000.
8. J. Duckett et al, "Professional XML schemas", Wrox Press Inc, 1st edition, 2001
9. Cattell R. et al, "Java 2 Platform, Enterprise Edition: Platform and Component Specifications", Addison-Wesley Pub Co, 2000.
10. Roman Ed et al., "Mastering Enterprise JavaBeans" 2nd Edition, Wiley Computer Publishing, 2002.
11. Enterprise Java Beans Specification version 2.1,Final Release, Sun Microsystems, 12 November 2003.
12. W. Yao, K. Moody, J. Bacon, "A model of OASIS Role-Based Access Control and its Support for Active Security", proceeding of SACMAT 2001, Chantilly, Virginia, USA, May 3-4, 2001.