Access-Control Language for Multidomain Environments

The XML Role-Based Access Control (X-RBAC) specification language addresses multidomain environments' policy-specification needs. X-RBAC is based on an extension of the widely accepted US National Institute of Standards and Technology role-based access-control (RBAC) model. In addition to allowing specification of RBAC policies and facilitating specification of timing constraints on roles and access requirements, X-RBAC provides a framework for specifying mediation policies in a multidomain environment where RBAC policies have been employed.

James B.D. Joshi
University of Pittsburgh

Rafae Bhatti, Elisa Bertino, and Arif Ghafoor
Purdue University

Recent advances in high-performance computing and networking technologies have fueled the growth of large-scale distributed applications. With the rapid proliferation of information technologies, security is a growing concern. Many studies show that unauthorized access, particularly by insiders, constitutes a major security problem for enterprise applications.[1] The issue is magnified in multidomain environments, in which multiple distributed organizations — each with its own security policy — interoperate.[2,3]

XML technology has emerged as the most promising approach for developing pragmatic security solutions for multidomain environments. XML allows uniform representation, interchange, sharing, and dissemination of information over heterogeneous environments.[4] The key challenge for securing an XML-based multidomain environment is developing access-control models that grant access based on environmental context (such as administrative domain or access time) and information content, object and subject type, profile, or qualifications, and facilitate easy integration of multiple security policies.[2]

Role-based access-control (RBAC) models show clear advantages over traditional discretionary and mandatory access-control models[2,5] with regard to these requirements. In particular, an RBAC approach allows uniform representation of diverse security policies and supports efficient access management. Our XML-based access-control policy (X-RBAC) specification language extends the US National Institute of Standards and Technology (NIST) RBAC model[6] with temporal constraints, role attributes, contextual conditions, a notion of role states, and preconditions for state transitions. X-RBAC provides a wide range of protection granularity for protected data and supports policy mapping in multidomain environments. Our approach allows access control at the element-level granularity of XML sources and enforces concept-level access control on huge document repositories.

Although researchers have aligned XML technologies with RBAC systems,[7–10] to the best of our knowledge, no one has investigated an XML-based

 Published by the IEEE Computer Society 1089-7801/04/$20.00

RBAC language for access management in multidomain environments. In addition, our model supports specification of policies in arbitrary multidomain environments.

Extended RBAC Model

Figure 1 is a simplified version of the NIST RBAC model, which comprises four key elements: users, roles, permissions, and sessions. System administrators can apply constraints to the assignment of users and permissions to roles and users' activation of roles in sessions. A user assigned to a role can activate the role in a session and acquire all the permissions assigned to it. We extend the NIST model with parameterized roles, context and content-based constraints, and XML-based policy components (see Figure 1) for specifying various elements of the extended-RBAC model.

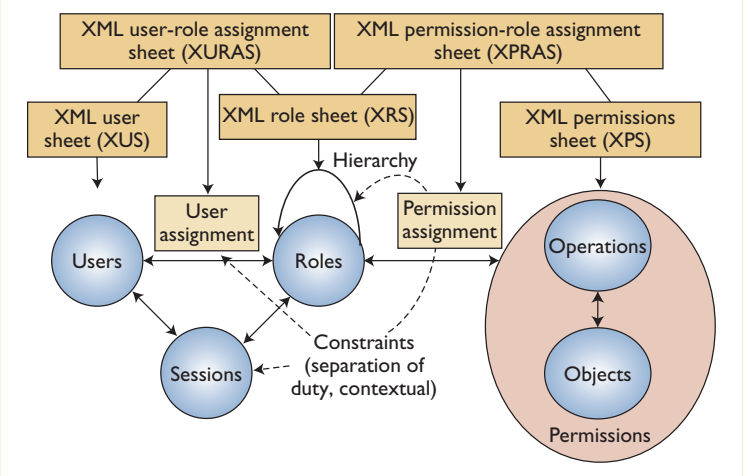

Figure 1. US National Institute of Standards and Technology role-based access control (RBAC) model and X-RBAC policy components. The X-RBAC framework adds policy components to define the extended RBAC elements: users and their credentials, roles with attributes and associated preconditions, permissions, and sessions. For example, XML user sheets (XUS) define users and their credentials, XML role sheets (XRS) specify a role's attributes and context-based constraints using these attributes.

Roles with Attributes

Underlying our extended RBAC model is the notion that roles can have associated temporal constraints, specifying when they can be used. Depending on the application semantics, all roles might not be available to all users at all times. Our model reflects this notion by associating different states with roles. Figure 2 shows the three potential role states:

- *Disabled* indicates that the role can't be activated in a session.
- *Enabled* indicates that the users authorized for the role at the time of the request can activate the role.
- *Active* implies that at least one user has activated the role.

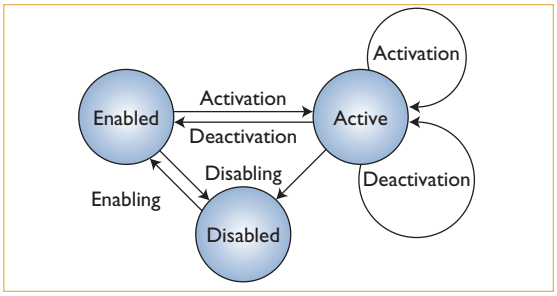

Figure 2. Role states. Disabled roles can't be activated in a session, whereas enabled roles can be activated by authorized users.

A role in enabled or active state transitions to disabled state if the system generates a role-disabling event. For instance, after a role stays enabled for a specified interval of time, the system generates the disabling event to disable it and prevents users from activating it. The model allows capturing preconditions that define how a role might change its state. Each precondition consists of logical conditions defined on elements of a role's parameter set. For instance, a role enabling precondition might simply involve checking whether the current time is an instant of the specified interval defined for that role. Similarly, a role activation precondition might simply involve checking whether the user's network domain matches some prespecified domain name. The model also allows specification of preconditions for controlling assignment of users and permissions to roles. For instance, a user assignment precondition may specify that a user might be assigned to a given role if the user's credential values satisfy the prespecified criteria.

Role-assignment parameters typically include attributes whose values:

- the system administrator assigns to a user together with the authorization to use the role (typically, these refer to prespecified organization-specific values) or
- the user provides and for which the user must have a certificate (typically, these refer to generic values an unknown user can present).

The attribute sets for the three types of precondi-

```
<PeriodicTimeExpr   pt_expr_id = "PT1"
          pt_begin = "2003-01-01"
          pt_end = "2003-12-31">
  <StartTimeExpr>
    <Day daySet = "Monday, Wednesday"/>
    <Hour hourSet = "9AM"/>
  </StartTimeExpr>
  <DurationExpr cal = "Hours" len = 12/>
</PeriodicTimeExpr>
```

Figure 3. Periodicity expression. The XML specification defines timing constraints. The example specifies intervals between 9 a.m. and 9 p.m. of every Monday and Wednesday in 2003.

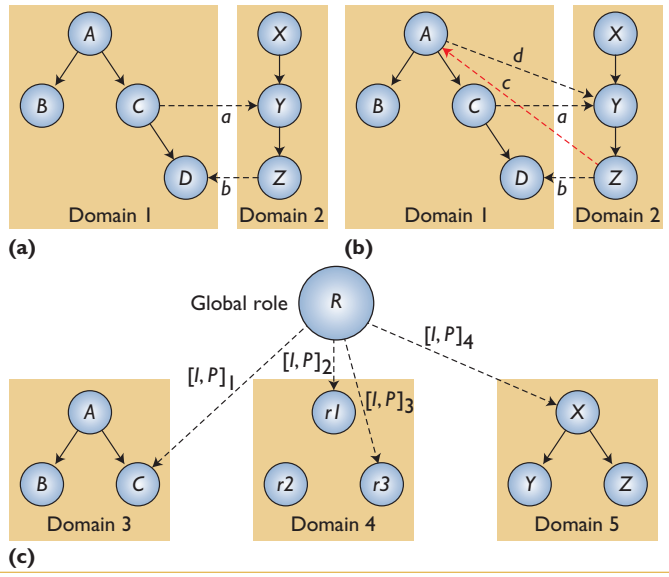

Figure 4. Example multidomain environments. A–D and X–Z are roles, with links indicating inheritance. (a) Links a and b indicate interdomain access between domains 1 and 2 such that users authorized for role C are also authorized for roles Y and Z. (b) Adding links c and d leads to a violation of the security principle, by extending authorization for role C to role A through inheritance. (c) Role mapping from global role R *to local roles in the three domains indicates what roles a user assigned to* R *may assume in the local domains in a federated environment.*

tions aren't disjoint; the same parameters can affect role enabling, assignment, and activation. The "Role Preconditions" sidebar illustrates the notion of a parameterized role and role preconditions.

Capturing Context and Content Information

In general, we can use parameterized roles to capture context-based access requirements by defining an appropriate set of parameters and predicates associated with them. Our specification framework captures time and location context.

Time. We use the periodic time expression represented by pairs of the form $[I, P]$ and *calendars* – that is, countable sets of contiguous intervals – to express timing constraints.[11] We write $C_a \subseteqq C_b$ if each interval of calendar C_b is covered by a finite number of intervals of C_a. P is a periodic expression denoting an infinite set of periodic time instants, and I = (`begin`, `end`) is an interval denoting the lower and upper bounds imposed on instants in P. Formally,

$$P = \Sigma_{i=1}^{n} \quad O_i.C_i \triangleright x.C_d,$$

where C_d, C_1, ..., C_n are calendars and $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$, $C_i \subseteqq C_{i-1}$ for $i = 2, ..., n$, $C_d \subseteqq C_n$, and $x \in \mathbb{N}$. The expression to the left of $\triangleright$ identifies the set of starting points of the intervals, and the expression to the right indicates each interval's duration in terms of calendar C_d. For example, $\{all.\text{Years} + \{3, 7\}.\text{Months} \triangleright 2.\text{Months}\}$ represents the set of intervals that start on the third and seventh months of every year and have two-month durations.[11] Figure 3 shows the XML specification for periodic time denoting every Monday and Wednesday between 9 a.m. and 9 p.m. in 2003. Calendars' Year, Month, and Week are by default assumed to be associated with *all* (for example, *all*.Year).

Location. A session parameter records the user domain associated with an access request to provide location-based access control. Additionally, X-RBAC lets us capture attributes that profile user activities – for example, login_time, login_date, and session duration. The system processes such information dynamically and incorporates it into access decisions.

Content. We allow content-based access control for XML document sources at four levels: conceptual, XML schema, XML instance, and XML element. We use a cluster-based approach to specify conceptual-level access control by grouping information content into concept clusters using a similarity-based function for content classification.[12]

RBAC Policy Specification Framework

Multidomain environments are evident in several emerging systems, most prominently in Web services and grid-based systems.[13,14] Web services typically appear in business-to-business applications in which service providers expose specific informa-

Role Preconditions

We use sets X_{en}, X_{as}, and X_{ac} to denote the elements of X associated with the three preconditions: X_{en} corresponds to role-enabling, X_{as} to role assignment, and X_{ac} to role activation.

For the role `DoctorInTraining` in a hospital, where X = {time instant (t), time duration (d), system load (l), user (u), role (x), certification (c)}, we apply the three preconditions.

Role-enabling precondition ($X_{en} = \{t, l\}$). The system enables the role `DoctorInTraining` if the following conditions hold:

1. Time instant t falls on every workday between 9 a.m. and 9 p.m.
2. System load l is low. Here, we assume that the system load characterizes the number of doctors and nurses currently on active duty.

Role assignment precondition ($X_{as} = \{c, t\}$). The system assigns Dr. Smith to `DoctorInTraining` if the following conditions hold:

1. Dr. Smith's certification c is valid. Here, c represents the certificate of eligibility provided by a certifying authority. We define a predicate *ValidCertificate(c)* for this purpose.
2. Time instant t falls on every Monday, Tuesday, and Wednesday between 9 a.m. and 9 p.m.

Role activation precondition ($X_{ac} = \{t, u, r, d\}$). An authorized user can activate `DoctorInTraining` if the following conditions hold:

1. Dr. Jones is on active duty as a supervisor. We can use predicate *active*(u, r) to test whether user u has activated role r. If active(u, r) returns true for u = "Dr. Jones" and r = `SupervisorDoctor`, this condition is satisfied.
2. Active duration d for Dr. Jones is less than or equal to two hours. We can define a predicate *ActiveDuration*(u, r, d) to check whether u has activated role r for duration d. We then use the predicate to check for u = "Jones," d = "2 hours."

The example also illustrates that we can use an attribute in more than one precondition type.

tion to clients, or automated transactions occur between two e-commerce applications. Grid-based systems can span an Internet-sized environment with heterogeneous systems distributed across multiple administrative domains.[13]

In a multidomain environment, such as those illustrated in Figure 4, the key security goal is to ensure that no violations occur during interdomain accesses. In particular, secure interoperation should enforce two principles[3]:

- *Autonomy*. An access that's permitted within an individual system must also be permitted under secure interoperation.
- *Security*. An access that isn't permitted within an individual system can't be permitted under secure interoperation.

Figure 4b illustrates a violation of the security principle. Let A, B, C, D, X, Y, and Z be roles and let the links indicate inheritance — that is, users authorized for role A are also authorized for roles B, C, and D; users authorized for C are also authorized for D, and so on. Let links a and b indicate interdomain accesses allowed between domains 1 and 2. Assuming the interdomain links also have inheritance semantics, users authorized for role C are also authorized for roles Y and Z. Clearly, links a and b uphold the two principles. If we add interdomain links c and d, as Figure 4b depicts, we violate the security principle: Users originally authorized for role C and not for role A are now authorized for role A because of the inheritance path from C to Y to Z to A.

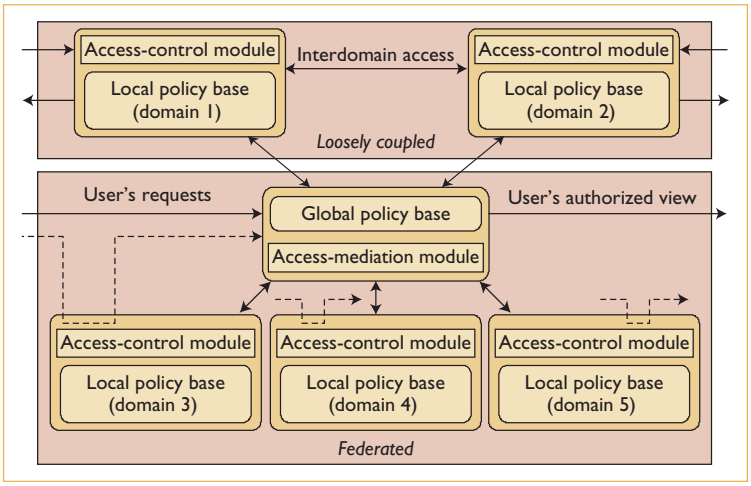

Figure 5. Architectural configurations in a multidomain environment. In a loosely coupled multidomain environment (top), systems agree to share information for a specified amount of time. In a tightly coupled, or federated environment, one system is designated as the global, or master domain, whereas the others are local. Arrows indicate the general flow of access requests. Dotted lines indicate that domains receiving access requests for information in other domains redirect the requests to the global layer.

Figure 5 depicts two architectural configurations characterizing a multidomain environment: loosely coupled and tightly coupled (or federated) multidomain environments. In the figure, domains

Table 1. Example role mapping in a federated system.

Mapped to→	Hospital 1 role *C*	Hospital 2 roles *r*1 and *r*2	Hospital 3 role *X*
R is `FederatedDoctor`	*C* = `DayDoctor` in PT_1 = $[I,P]_1$, representing every Monday and Wednesday	*r*1 = `DayDoctor` in PT_2 = $[I, P]_2$, representing every Tuesday and Thursday *r*2 = `EmergencyDoctor` in PT_3 = $[I, P]_3$, representing every Friday	*X* = `SupervisorDoctor` in PT_4 = $[I, P]_4$, representing every weekend

```
<!-- Policy Definition --> ::=
<XPolicy [policy_id = "(value)"]>
  <PolicyName> (name)</PolicyName>
  <!-- XML User Sheet -->
  <!-- XML Role Sheet-->
  <!-- XML Permission Sheet-->
  <!-- XML User-Role Assignment-->
  <!-- XML Role-Permission Assignment-->
  [<!-- Local Policy Definitions-->]
  [<!-- Policy Relationship Definitions-->]
</XPolicy>
```

Figure 6. X-RBAC policy specification format. The policy definition contains the optional policy id; a policy name; XML policy components related to users, roles, permissions, assignment of users and permissions to roles; and optional local policy definitions of subdomains.

3, 4, and 5 form a federated environment, whereas domains 1 and 2 form a loosely coupled environment. Multidomain environments can contain both loosely coupled and federated components.

Loosely Coupled Multidomain Environments

In a loosely coupled multidomain environment, independent systems dynamically come together to share information for a period of time. For example, a company that lets its consulting firms partially share information during a contractual period creates a loosely coupled environment with its consultant companies.

Two access-mediation approaches are available for these environments. The first uses a predefined set of role mappings to mediate interdomain accesses. This approach requires the constituent systems to indicate the level of sharing they want to allow and to establish a consistent set of mediation rules for interdomain accesses. The second approach uses certificates to map unknown principals to predefined roles. This approach is suitable for environments like the Internet, in which anyone presenting required credentials or role attribute values that satisfy role preconditions gets access. Typically, this method relies on a trust-management infrastructure.[15] Either approach can involve a trust-negotiation phase.

Federated Multidomain Environments

In a federated multidomain environment, one system is typically designated master; the others are local domains. The master mediates accesses to individual systems through a global policy. For example, we can view a digital government as a federated system attempting to provide a set of services by federating several government units.[2] Such systems can also characterize merged organizational systems in which the policies are integrated. We can consider a grid a federated system in which donor systems join by submitting their local policies.[13,14] Typically, the global policy maps to local policies. For instance, a global role can map to various local roles in individual domains. Figure 4c shows the mapping of a global role R_1 to the local role *C* in domain 3, r_1 and r_3 in domain 4, and *X* in domain 5. Table 1 depicts such a mapping specific to a healthcare application environment.

Using this mapping, we can assign a doctor who needs to be cross-appointed to different hospitals at different times — for instance, to the `FederatedDoctor` role between 9 a.m. and 6 p.m. on Monday through Saturday. This means that during those hours, Dr. Smith can assume the role `DayDoctor` in hospital 1 on Mondays and Wednesdays, and in hospital 2 on Tuesdays and Thursdays; `EmergencyDoctor` in hospital 2 on Fridays; and `SupervisorDoctor` in hospital 3 on Saturdays.

XML-Based Specification Language for RBAC Model

Our XML specification framework, X-RBAC, expresses RBAC policies as well as mediation policies in both loosely coupled and federated multidomain environments.

Figure 6 shows the XML syntax for policy specification. Our policy-specification components (described in the following section) correspond to various RBAC components depicted in Figure 1. With X-RBAC, the system administrator can specify integrated policies by including other policy definitions as components through `<!— Local Policy Defin-`

itions —>, thus supporting the dynamic creation of articulated and complex multidomain policies. A constituent policy can be a local policy in a federated system or a partner domain policy in a loosely coupled environment. We include local policy definitions or simply refer to them using local policy IDs. If we define local policies, we must also define the set of relationships between global and local policies <!— Policy Relationship Definitions—>.

Because of size limitations, details on the syntax of the specification language are presented elsewhere.[16] In this article, we present policy specification examples using the language.

X-RBAC Policy Specification

We use several basic specification components.

XML user sheets (XUS) define users and their credential types. Credentials defining user qualifications and profile information are necessary when unknown users will have system access. The system checks whether the presented credential values satisfy assignment preconditions for the requested role. Figure 7 shows an instance of XUS that defines a user and his credentials.

A credential type definition specifies the attribute list associated with a credential type. Attributes can be mandatory (mand) or optional (opt). Consider the following user credential based on a general credential expression of the form ((cred_type_name cred_type_id), cred_expr), where cred_type_id is a unique credential type identifier and cred_expr is a set of attribute-value pairs:

```
((Nurse, "C100"), {(user_name, "John",
mand), (age, 30, opt), (level, fifth, mand)}).
```

User definitions can simply define the user_name and user_id, or also specify the assigned credentials the user might carry. In Figure 7, the MaxRoles tag indicates the maximum number of roles the user can be assigned to.

We define permissions for a given system in terms of objects and associated operations. Figure 8 is an example XML permission sheet instance; in it, perm_id is a unique permission identifier. An object can represent a cluster, schema, instance document, or document element. We use object types to distinguish them. The system administrator assigns IDs to clusters, schemas, and documents. XML Path Language (XPath) expressions specify elements within a given XML document. A user with access privileges to a cluster also has access to all schemas and instance documents belonging to the cluster. Similar semantics apply to schema and their elements and instances. Propagation options indicate whether a permission propagates down the object hierarchy.[4]

```
<XUS>
  <User u_id = "u1">
    <UserName> JSmith </UserName>
    <CredType  c_type_id = "C100">
      <CredExpr>
        <FName>John</FName>
        <LName>Smith</LName>
        <age>30</age>
        <level>5</level>
      </CredExpr>
    </CredType>
    <MaxRoles> 2 </MaxRoles>
  </User>
</XUS>
```

Figure 7. An example XML user sheet showing the definition of a user, John Smith, his credential information, such as age and level, and the maximum number of roles he can assume.

In Figure 8, permission P1 allows a read operation on all documents within the scope of cluster CL100 with the default propagation option (that is, "no prop"). Similarly, permission P2 allows all operations on all document instances conforming to schema XS101, and P3 allows all operations on document instance XI100, both with the default propagation option. Lastly, permission P4 allows a navigate operation on the XML name element, also with the default propagation option.

A role can have associated enabling, assignment, and activation preconditions, which we define using the <EnabCondition>, <AssignCondition>, and <ActivCondition> tags. For enabling and disabling preconditions, we use the periodic time expression (Figure 3) as a condition. X-RBAC language includes a generic syntax for expressing an arbitrarily complex logical expression.

A role definition can specify hierarchy relations using the <Junior> and <Senior> tags, and express role cardinality using the <Cardinality> tag. To specify separation-of-duty (SoD) constraints, we construct a role set and specify a cardinality stating how many roles from the set we can assign to a user (Static SoD, or SSD) or a user can activate (Dynamic SoD, or DSD).

Figure 9 shows an XML instance document describing SpecialDoctor and Database Administrator roles along with the corresponding SSD and DSD role sets. Accordingly, SpecialDoctor

```
<XPS>
  <Permission perm_id = "P1">
    <Object type = "Cluster"   id = "CL100">
      EyeDisease
    </Object>
    <Operation> read </Operation>
  </Permission>
  <Permission perm_id = "P2">
    <Object type = "Schema"  id = "XS101">
      PatientEyeReport
    </Object>
    <Operation> all </Operation>
  </Permission>
  <Permission perm_id = "P3">
    <Object type = "Instance"  id = "XI100">
      EyeReportForJoe
    </Object>
    <Operation> all </Operation>
  </Permission>
  <Permission perm_id = "P4">
    <Object type = "Element"  id = "XE100">
      EyeColor
    </Object>
    <Operation> navigate </Operation>
  </Permission>
</XPS>
```

Figure 8. An instance of XML permission sheet describing permissions as permitted operations over various object types, such as cluster CL100 and XML schema document XS101.

```
<XRS>
  <Roles>
   <Role role_name="SpecialDoctor">
     <EnabCondition pt_expr_id= "PT1">
        <LogicalExpr>
          <Predicate>
            <Operator>eq</Operator>
            <FuncName>isActive</FuncName>
            <NameParam type=role>
               SupervisorDoctor</NameParam>
            <RetValue>true</RetValue>
          </Predicate>
        </LogicalExpr>
     </EnabCondition>
     <Junior> Resident </Junior>
     <Cardinality>8</Cardinality>
   </Role>
   <Role role_name="DBA"/>
  </Roles>

  <SSDRoleSet ssd_id = "SSD1"
       ssd_cardinality = "1">
    <SSDRole>Nurse</SSDRole>
    <SSDRole>SpecialDoctor</SSDRole>
    <SSDRole>Dispenser</SSDRole>
    <SSDRole>DBA</SSDRole>
  </SSDRoleSet>

  <DSDRoleSet dsd_id = "DSD1"
       dsd_cardinality = "2">
    <DSDRole>DBA</DSDRole>
    <DSDRole>Accountant </DSDRole>
    <DSDRole>Cashier</DSDRole>
  </DSDRoleSet>
</XRS>
```

Figure 9. An instance of an XML role sheet describing two roles — `SpecialDoctor` and `DBA` — with their corresponding static (`SSD1`) and dynamic (`DSD1`) separation-of-duty role sets. The `SpecialDoctor` role is enabled in the time interval defined by periodic time expression PT1 (that is, between 9 a.m. and 9 p.m. every Monday and Wednesday in 2003) and if the `SupervisorDoctor` role is active.

belongs to the SSDRoleSet identified by `SSD1` with cardinality 1, and hence we can't assign users to more than one role from this set. Similarly, the `DBA` role belongs to the DSDRoleSet identified by `DSD1` with cardinality 2, and hence authorized users can activate no more than two roles at the same time.

The system administrator uses an *XML user-role assignment sheet* (XURAS) to assign users to a role by associating with it the required credentials, and an *XML permission-role assignment sheet* (XPRAS) to assign permissions to a role. Figure 10 shows an XURAS instance. The example associates a set of credentials with the `SpecialDoctor` role. It states that any user with the credential type `Nurse` can be assigned to the `SpecialDoctor` role only if the user's level is greater than 5 and age is less than 80.

The assignment of permissions to corresponding roles reflects the policy specifications at the conceptual, schema, instance, and element levels in an XPRAS. Conceptual-level access control uses roles related to concepts, as Figure 11 illustrates. In Figure 11, the mapping identified by `PRM1` associates the `EyeDoctor` role with permission `P1`, which refers to a concept object cluster (see Figure 8). In

```
<XURAS>
  <URA ura_id="URA1" role_name =
       "SpecialDoctor ">
  <AssignUsers>
    <AssignUser user_id = "ANY">
    <AssignCondition pt_expr_id = "PT1">
      <LogicalExpr op = "AND">
        <Predicate>
          <Operator> gt </Operator>
          <ParamName>level</ParamName>
          <RetValue>5</RetValue>
        </Predicate>
        <Predicate>
          <Operator> lt </Operator>
          <ParamName>age</ParamName>
          <RetValue>80</RetValue>
        </Predicate>
      </LogicalExpr>
    </AssignCondition>
    </AssignUser>
  </AssignUsers>
  </URA>
</XURAS>
```

Figure 10. Instance of an XML user-role assignment sheet. This XURAS defines the `SpecialDoctor` *role and an assignment precondition on user credentials and time.*

this case, an `EyeDoctor` role is authorized to read all the documents in the cluster with ID CL100.

We can similarly protect XML schemas, document instances, and the elements within by associating them with corresponding roles. For instance, in Figure 11, the mapping identified by `PRA2` associates the `DBA` role with permissions `P2` and `P3`, which refer to a schema object and an instance document, respectively (see Figure 8). In this case, the `DBA` role is authorized to read, write, and navigate all instance documents conforming to the schema ID `XS101` and the instance document XI100. Similarly, the mapping identified by `PRA3` associates the `Dispenser` role with permission `P4` referring to the `Name` element. Hence, the `Dispenser` role is authorized only to navigate the `Name` element in all conforming instance documents.

```
<XPRAS>
  <PRA pra_id="PRA1" role_name="EyeDoctor">
    <AssignPermission perm_id = "P1">
  </PRA>
  <PRA pra_id="PRA2" role_name="DBA">
    <AssignPermission perm_id = "P2">
    <AssignPermission perm_id = "P3">
  </PRA>
  <PRA pra_id="PRA3" role_name="Dispenser">
    <AssignPermission perm_id = "P4">
  </PRA>
</XPRAS>
```

Figure 11. Instance of an XML permission-role assignment sheet. This shows the assignment of the `EyeDoctor` *role to permission P1, allowing the* `EyeDoctor` *role to read all documents in cluster CL100.*

Mediation Policies

As Figure 12 shows, we can include local policy definitions within a policy definition using `<!-Local Policy Definitions –>`. Local policies are defined using the syntax for `<!– Policy Definition –>` shown in Figure 6 and are placed between the tags `<XLPD>` and `</XLPD>`. Each policy can be a global policy over a set of locally dominated domains. In our specification, *mediation policies* – that is, policies that indicate how roles from a policy are related to the roles in the local policies – are specified by defining relationship definitions using `<!–Policy Relationship –>`. The policy specification uses the following scoping rule:

```
<!–Local Policy  Definitions --> ::=
<XLPD>
       [<!– Policy Definition -->]+
</XLPD>

<!–Policy Relationship  Definitions -->::=
<XPRD>
       [<!--Policy Relationship-->]+
</XPRD>
```

Figure 12. Definitions of local policies and mapping relations. The specification format shows that a policy can contain an arbitrary number of local policies. The relationship definitions capture the relation between the components of the policy and components of its local policies to establish mediation policies.

If a policy *P* becomes a local policy of a higher level policy, *P*'s local policy definitions and the policy relations are unknown to the higher level policy.

This scoping rule indicates that only the locally dominated policy entities within a global policy def-

inition are visible, rather than the entities of the local policies' constituent domains. This abstraction simplifies metapolicy construction. However, if the higher-level policy management must oversee the overall federation consistency, we might need to relax this rule. Figure 13 shows the XML syntax for defining policy relationships. We can map each external role onto several local roles belonging to the same or different local domains. For each mapping, we specify a condition. We include the local roles onto which an external role is mapped in the local policy definitions. Figure 14 is the specification of the interdomain role mapping defined in Figure 4c.

```
<!-- Policy Relationship --> ::=
  <XPR xpr_id = (id) [pt_expr_id = (id)]>
    <InterDomainMapping [idMap_id = (id)] >
      <RoleMapping>
       [<Mapping Definition>]+
      </RoleMapping>
    </ InterDomainMapping >
  </XPR>

  <Mapping Definition>::=
    <MappedRole>
      <Role[role_id = (id)] [policy_id = (id)]>
       (name)</Role>
       [<Mapped(To|From) Definition>]+
    </MappedRole>

    <Mapped(To|From) Definition>::=
    <Mapped(To|From)>
      <Role [role_id=(id)] [policy_id = (id)]>
       (name)</Role>
      <MappingCondition [pt_expr_id = (id)>
        [<!--LogicalExpression-->]
      </MappingCondition>
    </Mapped(To|From)>
```

Figure 13. Policy relation syntax. This syntax allows specifying mapping between a policy's roles to roles in its local policies, and indicates conditions that must be satisfied for the mapping to be valid.

We use the same structure to capture mediation policies for loosely coupled systems. In this case, each local policy definition comprises a part of the domain policy of the partner domain. For example, consider domains 1 and 2 in Figure 4a, with policies 1 and 2. In the policy 1 specification, some entities of policy 2 are known (specifically, roles *Y* and *Z*) and appear in the `<!– Local Policy Definitions –>` section. Similarly, some entity definitions of policy 1 (that is, roles *C* and *D*) will appear in policy 2 as local policy definitions. Role-to-role mappings can be from one domain to the other, as Figure 4a shows, or bidirectional. To capture mapping direction, we include the `<MappedFrom>...</MappedFrom>` syntax similar to the `<MappedTo>...</MappedTo>` syntax. Figure 15 shows snapshots of the relationship definitions for the role-to-role mapping.

Policy-Integration Challenges

Several issues – such as semantic heterogeneity and policy consistency – pose considerable challenges in multidomain environments. To manage semantic heterogeneity and integration of multiple heterogeneous policies in an XML-integrated mul-

```
<XPR xpr_id =" XPRg">
  < InterDomainMapping [idMap_id = "IDMg"]>
    <RoleMapping>
      <MappedRole>
        <Role policy_id = "Global">
          FederatedDoctor</Role>
      </MappedRole>
      <MappedTo>
        <Role policy_id = "Policy3">
          DayDoctor</Role>
        <MappingCondition pt_expr_id = "PT1" />
      </MappedTo>
      <MappedTo>
        <Role policy_id="Policy4">
          EmergencyDoctor</Role>
        <MappingCondition pt_expr_id = "PT1 />
      </MappedTo>
      <MappedTo>
        <Role policy_id = "Policy5">
          SupervisorDoctor</Role>
        <MappingCondition pt_expr_id = "PT1" />
      </MappedTo>
    </RoleMapping>
  </ InterDomainMapping >
</XPR>
```

Figure 14. Policy relation specification for the example in Figure 4c. The example shows that the global role `FederatedDoctor` *is mapped to the* `DayDoctor` *role of Policy3 of Domain 3 provided that the current time is in the intervals specified by the periodic time expression PT1 of Figure 1.*

```
<XPolicy policy_id = “Policy 1”>
  <!--Policy Definition-->
  <XPR pxr_id = “XPR1”>
    < InterDomainMapping idMap_id = “IDM1”>
      <RoleMapping>
        <MappedRole>
          <Role policy_id = “Policy1”> C
          </Role>
        </MappedRole>
        <MappedTo>
          <Role policy_id = “Policy2”> Y
          </Role>
        </MappedTo>
      </RoleMapping>
      <RoleMapping >
        <MappedRole>
          <Role policy_id = “Policy1”>D
          </Role>
        </MappedRole>
        <MappedFrom>
          <Role policy_id = “Policy2”> Z
          </Role>
        </MappedFrom>
      </RoleMapping>
    </ InterDomainMapping >
  </XPR>
</XPolicy>
```

(a)

```
<XPolicy policy_id = “Policy2”>
  <!--Policy Definition-->
  <XPR xpr_id = “XPR2”>
    < InterDomainMapping idMap_id = “IDM2” >
      <RoleMapping>
        <MappedRole>
          <Role policy_id = “Policy2”> Z
          </Role>
        </MappedRole>
        <MappedTo>
          <Role policy_id = “Policy1”> D
          </Role>
        </MappedTo>
      </RoleMapping>
      <RoleMapping >
        <MappedRole>
          <Role policy_id = “Policy2”> Y
          </Role>
        </MappedRole>
        <MappedFrom>
          <Role policy_id = “Policy1”> C
          </Role>
        </MappedFrom>
      </RoleMapping>
    </ InterDomainMapping >
  </XPR>
</XPolicy>
```

(b)

Figure 15. Policy relation specification for example in Figure 4a: (a) domain 1 and (b) domain 2. The specification shows that domain 1's policy includes domain 2's policy as its local policy and domain 2's policy includes domain 1's policy as its local policy. The relationship definitions for domain 1 capture the mapping from external role C to local role Y and local role Z to external role D in domain 1's policy.

tidomain environment,[2,7] ontologies and approaches for semantic integration of heterogeneous database schema can be useful. Our policy-integration methodology for multidomain environments has four phases:

- *Pre-integration* deals with representation and organization of semantic information about each domain's policy entities that helps resolve semantic differences. This phase might involve a data-dictionary-based approach or a common ontology.
- *Policy comparison* involves semantic-conflict detection, including naming conflicts among domain roles or structural conflicts among role hierarchies. Information obtained in the pre-integration phase facilitates this detection. We need techniques for detecting semantic conflicts, automatically or semiautomatically.
- *Policy conformance* deals with resolving semantic and rule conflicts. We need automatic techniques to synthesize mediation policies when security violations such as one in Figure 4b occur.
- *Merging and restructuring* deals with needed local-policy readjustments for obtaining a consistent merged policy after removing inconsistencies.

This process can be iterative. In particular, removing inconsistencies can entail considerable restructuring and refinement of the mediation policies. We perform this iteration mainly during the merging and restructuring and policy-conformance phases.

Future Work

We plan to extend our work in several directions. Because the XUS maintains a lot of user data, a user privacy provision is highly desirable, and we plan to pursue X-RBAC extensions to provide privacy preferences. We'll also extend our framework to allow interoperation of our mechanism with single-sign-on mechanisms and apply it to secure-

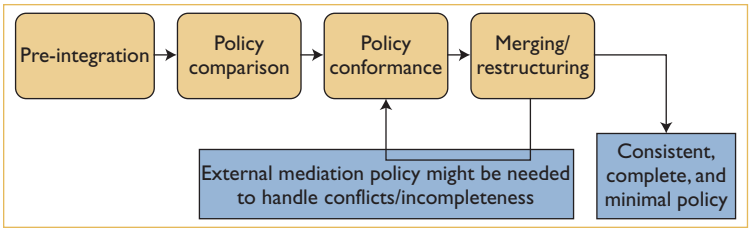

Figure 16. Policy-integration phases for multidomain environments. The iterative process identifies and resolves semantic and rule conflicts among local policies, making necessary adjustments to achieve a consistent merged policy. Handling conflicts or incompleteness might require an external mediation policy.

ly compose interoperable Web services. Furthermore, we plan to extend the X-RBAC language to include the full set of temporal constraints introduced in the generalized temporal role-based access control (GTRBAC) model.[11]

References

1. R. Power, *"Tangled Web": Tales of Digital Crime from the Shadows of Cyberspace*, Que/Macmillan Publishing, 2000.
2. J.B.D. Joshi et al., "Digital Government Security Infrastructure Design Challenges," *Computer*, vol. 34, no. 2, Feb. 2001, pp. 66-72.
3. L. Gong and X. Qian, "Computational Issues in Secure Interoperation," *IEEE Trans. Software and Eng.*, vol. 22, no. 1, Jan. 1996, pp. 43-52.
4. E. Bertino, S. Castano, and E. Ferrari, "Securing XML Documents with Author X," *IEEE Internet Computing*, vol. 5, no. 3, May/June 2001, pp. 21-31.
5. S.L. Osborn, R. Sandhu, and Q. Munawer, "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Trans. Information and System Security*, vol. 3, no. 2, Feb. 2000, pp. 85-106.
6. D. Ferraiolo et al., "The NIST Model for Role-Based Access Control: Towards a Unified Standard," *ACM Trans. Information and System Security*, vol. 4, no. 3, Aug. 2001, pp. 224-274.
7. N.N. Vuong, G.S. Smith, and Y. Deng, "Managing Security Policies in a Distributed Environment Using Extensible Markup Language (XML)," *Proc. Symp. Applied Computing*, ACM Press, 2001, pp. 405-411.
8. T. Moses, ed., "OASIS eXtensible Access Control Markup Language (XACML) Version 1.1," committee specification, 24 July 2003; www.oasis-open.org/committees/xacml/repository/cs-xacmlspecification-1.1.pdf.
9. A. Anderson, ed., "XACML Profile for Role-Based Access Control (RBAC)," OASIS Access Control TC committee draft 01, 13 Feb. 2004,
10. A. Kern, "Advanced Features for Enterprise-Wide Role-Based Access Control," *Proc. Ann. Computer Security Applications Conf.*, IEEE CS Press, 2002, pp. 333-343.
11. J.B.D. Joshi et al., "Generalized Temporal Role-Based Access Control Model," *IEEE Trans. Knowledge and Data Eng.*, accepted for publication.
12. R. Bhatti et al., "XML-Based Specification for Web-Services Document Security," *Computer*, vol. 37, no. 4, Apr., 2004, pp. 41-49.
13. F. Azzedin and M. Maheswaran, "Towards Trust-Aware Resource Management," *Proc. 2nd IEEE/ACM Int'l Symp. Cluster Computing and the Grid* (CCGrid 02), IEEE CS Press, 2002, pp. 452-457.
14. L. Pearlman et al., "A Community Authorization Service for Group Collaboration," *Proc. IEEE 3rd Int'l Workshop Policies for Distributed Systems and Networks*, IEEE Press, 2002, pp. 50-59.
15. M. Blaze et al., "The KeyNote Trust-Management System, version 2," IETF RFC 2704, Sept. 1999; www.ietf.org/rfc/rfc2704.txt.
16. J.B.D. Joshi et al., *X-RBAC: An Access-Control Language for Multi-Domain Environments*, tech. report 2004-46, CERIAS, 2004; www.cerias.purdue.edu/tools_and_resources/bibtex_archive/2004-46.pdf.

James B.D. Joshi is an assistant professor at the University of Pittsburgh and a coordinator of the Laboratory of Education and Research on Security Assured Information Systems (LERSAIS). His research interests include information systems security and distributed multimedia systems. He has a PhD in computer engineering from Purdue University. He is a member of the ACM and the IEEE. Contact him at jjoshi@mail.sis.pitt.edu.

Rafae Bhatti is a PhD candidate in the Electrical and Computer Engineering Department at Purdue University. His research interests include information systems security and distributed systems. He has an MS in computer engineering from Purdue University. He is a student member of the IEEE. Contact him at rafae@purdue.edu.

Elisa Bertino is a professor in the Computer Sciences Department at Purdue University and the research director of CERIAS. Her research interests include security, database systems, object technology, and Web-based information systems. Bertino received a PhD in computer science from the University of Pisa. She is a fellow of the IEEE and the ACM. Contact her at bertino@cerias.purdue.edu.

Arif Ghafoor is a professor in the Electrical and Computer Engineering Department and director of the Distributed Multimedia Systems Laboratory at Purdue University. His research interests include database security, parallel and distributed computing, and multimedia information systems. He has a PhD in electrical engineering from Columbia University. He is a Fellow of the IEEE. Contact him at ghafoor@dynamo.ecn.purdue.edu.