CERIAS Tech Report 2004-46

X- RBAC : AN ACCESS CONTROL LANGUAGE FOR MULTI-DOMAIN ENVIRONMENTS

by James Joshi, Rafae Bhatti, Elisa Bertino, Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

X- RBAC
An Access Control Language for Multi-domain Environments

James B. D. Joshi[1],
Rafae Bhatti[2],
Elisa Bertino[2],
Arif Ghafoor[2]

[1]University of Pittsburgh, Pittsburgh, PA

[2]CERIAS & Purdue University, West Lafayette, IN

Contact Author: jjoshi@mail.sis.pitt.edu

Abstract

A multi-domain application environment consists of distributed multiple organizations, each employing its own security policy, allowing highly intensive inter-domain accesses. Ensuring security in such an environment poses several challenges. XML technologies are being perceived as the most promising approach for developing pragmatic security solutions for such environments because of the integration and interoperation framework they provide. In this paper, we highlight these challenges and propose an XML-based access control specification language called X-RBAC that addresses policy specification needs of a multi-domain environment. Our specification language is based on an extension of the widely accepted NIST RBAC model. X-RBAC allows specification of RBAC policies and facilitates specification of timing constraints on roles as well as context and content-based access requirements. Furthermore, it provides a framework for specifying mediation policies in a multi-domain environment where RBAC policies have been employed.

Keywords: XML, RBAC, Access Control Policy, Security, Multi-domain

1. Introduction

Tremendous growth of large-scale distributed applications has been fueled by the recent advances in high-performance computing and networking technologies. With rapid proliferation of IT technologies, security is becoming a major concern. Many studies show that unauthorized access, in particular by *insiders*, constitute a major security problem for enterprise applications [Pow00] highlighting the need for robust access control systems. This problem is magnified in a multi-domain environment where distributed multiple organizations, each employing its own security policy, interoperate with each other [Jos01, Gon96]. Such multi-domain environments are already a reality, as can be seen in most Internet-based applications, digital governments, and healthcare systems [Jos01]. EXtensible Markup Language (XML) technology has emerged as the most promising approach for developing pragmatic security solutions for such environments as it allows uniform representation, interchange, sharing and dissemination of information over heterogeneous environments [Ber01]. The key challenge for securing an XML-based multi-domain environment is developing access control models with the following capabilities:

1. *Context and content-based access*: Access may need to be restricted based on context such as users' administrative domains, time of access, or an environmental state. Access may also be restricted based on information content. For example, in the healthcare industry, selective content-based access to patient information should be given to physicians and insurance providers.
2. *Heterogeneity of subjects and objects*: Object heterogeneity may exist in the form of different types of media, concepts, or knowledge embodied in the information being protected. For instance, in a digital library, it may be desirable to exercise access control for high-level concepts instead of individual objects. Furthermore, information content can evolve with time, introducing scalability problems in privilege management. Subject heterogeneity implies that users have diverse activity profiles and qualifications that may not be known *a-priori*, making access management difficult.
3. *Policy heterogeneity management*: Each domain of a multi-domain environment can have its own security policy, and the integration of these local policies entails various challenges including reconciliation of semantic differences between local policies, secure interoperability, containment of risk propagation and policy management [Jos01]. In such an environment, uniform representation of policies of individual domains is desirable, and meta-policies are needed to mediate accesses across domain boundaries [Jos01]. Of particular importance in a multi-domain environment are the timing constraints. For instance, an organization may decide to interlink its information system with those of its consulting firms for a period of time defined in their contracts. When the contract expires, the systems should be de-linked to make sure that the organization's information assets are no more accessible to the consulting firms.

With regards to these requirements, Role Based Access Control (RBAC) models show clear advantages over traditional discretionary and mandatory access control models [Jos01, Osb00]. In particular, an RBAC approach allows uniform representation of diverse security policies and supports efficient security management.

In this paper, we propose an XML-based access control policy specification language that incorporates above mentioned capabilities. We extend the NIST RBAC model with temporal constraints, role attributes, contextual conditions, a notion of role states, and pre-conditions for state transitions. The proposed specification language provides a wide range of protection granularity for protected data and supports policy mapping in multi-domain environments. Our approach allows access control at the element-level granularity of XML sources, and provides the capability to enforce concept-level access control on huge document repositories. To the best of our knowledge, an XML-based RBAC language for access management in multi-domain environments has not been previously investigated. An XML based approach to specify enterprise RBAC policies has been reported in [Vuo01]. The OASIS XACML allows specification of context-based access specification and RBAC profiles [Urla, Urlb]. Kern *et. al.* present an Extended RBAC model for enterprise-wide control [Ker02]. However, they do not include the fine-grained notion of role states, role pre-conditions, and temporal constraint expression proposed here. Furthermore, our model provides support for specification of policies of arbitrary multi-domain environments. Keromytis *et. al.* propose the STRONGMAN architecture to support policy-based management of multi-domain systems. However, it uses KeyNote as low level policy language and does not support RBAC. STRONGMAN architecture provides scalability and can supplement the scheme we propose. Bonatti *et. al.* propose a policy composition algebra that can supplement the XML integration framework proposed here [Bon00].

2. Extension of RBAC Model with Context and Content information

The NIST RBAC model is comprised of four key elements, namely, *users*, *roles*, *permissions* and *sessions*. Figure 1.1(a) depicts the simplified version of the NIST RBAC model. Constraints can be applied to the assignment of user and permissions to roles, and activation of roles by users in sessions. The extensions proposed in this paper include role states, parameterization of roles, and inclusion of context information, mainly *time* and *location*, and content-based access control. Figure 1.1(a) depicts the policy components for defining various elements of our RBAC model, e.g., XML User Sheet (XUS) is used to define users and their properties. These will be presented in detail in later sections.

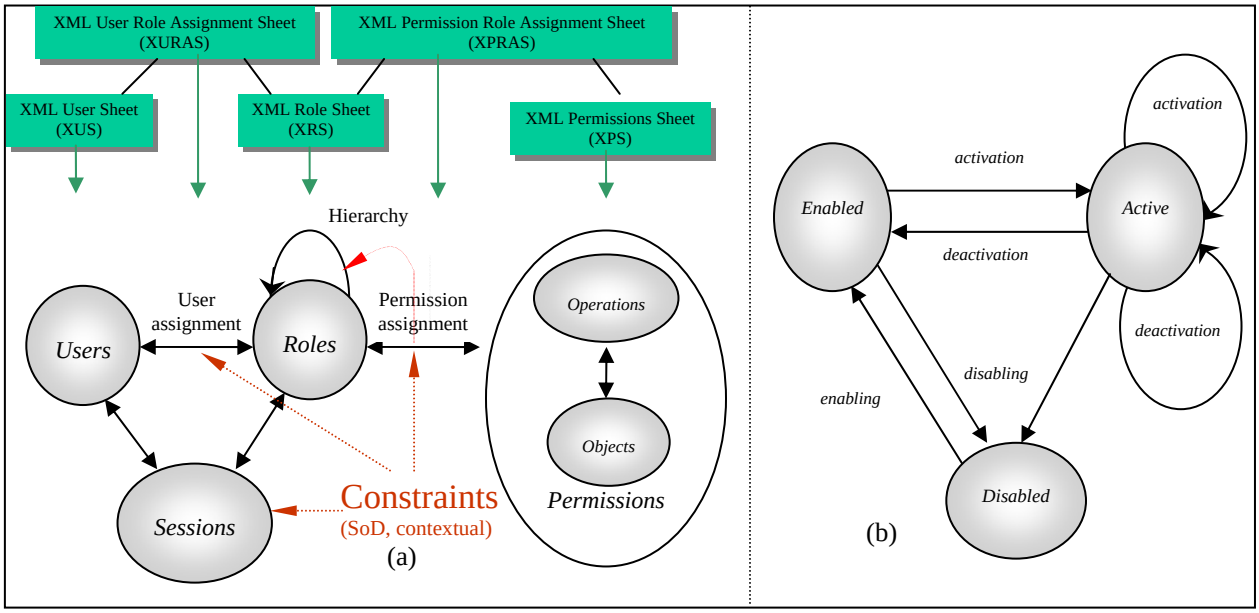

Figure 1.1 (a) The NIST RBAC Model and Policy Components (b) States of a Role

2.1 Roles with Attributes

An important notion underlying our RBAC model is that roles may have some associated temporal constraints, specifying when roles can be used. Depending on the application semantics, not all roles may be available to all users at any time. This notion is reflected in our model by associating different states with roles. Figure 1.1 (b) shows the three states that a role can be in: *disabled*, *enabled* and *active*. The *disabled* state indicates that the role cannot be activated in a session. A role in the *disabled* state can be enabled. The *enabled* state indicates that users authorized for the role at the time of the request may activate the role. A role in the *active* state implies that there is at least one user who has activated the role. A role in the *enabled* or *active* state transitions to the *disabled* state if a disabling event occurs. We define the following three types of preconditions to specify conditions on which a role can change its state:

1. *Role enabling/disabling precondition* that needs to be satisfied for the role to be enabled.
2. *Role assignment/deassignment precondition* that needs to be satisfied to assign a user to the role.
3. *Role activation/deactivation precondition* that needs to be satisfied before an authorized user can activate a role.

Each precondition consists of logical conditions defined on elements of a role's parameter set, say X. We use sets X_{en}, X_{as}, and X_{ac} to denote the elements of X that are associated with the three preconditions. X_{en} consists of parameters on which role *enabling* depends. Set X_{as} contains parameters on which role assignments depend. X_{as} typically includes:

- attributes whose values are assigned to a user together with the authorization to use the role; these typically refer to pre-specified organization specific values; or
- attributes whose values must be provided by the user and for which a certificate can be required; these typically refer to generic values that any previously unknown user may present.

Set X_{ac} represents the attributes related to activation of roles [Jos02]. Note that the three sets are not disjoint; that is, the same parameters may affect enabling, assignment and activation of a role. Table 1 illustrates the notion of a parameterized role and role preconditions for role DoctorInTraining in a hospital and X ={*time instant* (*t*), *time duration* (*d*), *system load* (*l*), *user* (*u*), *role* (*x*), *certification* (*c*)}.

Table 1. Example role preconditions

Role enabling precondition: (X_{en} = {*t, l*}): DoctorInTraining is enabled if conditions 1 and 2 hold:
1. *t* is a time instant that falls under *every workday between 9am–9pm*
2. *system load l* is low. Here, we assume that the *system load* characterizes how many doctors and nurses are currently in active duty
Role assignment precondition: (X_{as} = {*c, t*}): Dr. Smith can be assigned to DoctorInTraining if conditions 1 and 2 hold:
1. Dr. Smith's *certification c* is valid. Here, *c* represents the certificate of eligibility provided by a certifying authority. A predicate *ValidCertificate*(*c*) can be defined for this purpose.
2. *t* is a time instant that falls under *every Mondays, Wednesdays and Tuesdays between 9am–9pm,*
Role activation precondition: (X_{ac} = {*t, u, r, d*}): An authorized user can activate DoctorInTraining if conditions 1 and 2 hold:
1. Dr. Jones is on active duty as a supervisor. Predicate *active*(*u, r*) can be used to test if user *u* has activated role *r*. If *active*(*u, r*) returns *true* for *u* = "Dr. Jones" and *r* = SupervisorDoctor, then this condition is satisfied,
2. Active duration *d* for Dr. Jones is less than or equal to 2 hours. We can define a predicate *ActiveDuration*(*u*, r, *d*) to check if *u* has activated role *r* for duration *d*. It is then used to check for *u* = "Jones", *d* = "2 Hours".

2.2 Capturing Context and Content Information

In general, a parameterized role can be used to capture any type of context-based access requirements by defining an appropriate set of parameters and predicates associated with them. Below, we elaborate on how our specification framework capture *time* and *location* context.

```
<!-- Periodicity Expression -->
<PeriodicTimeExpr pt_expr_id = "PT1">
  <StartTimeExpr >
    <Day daySet = "Monday,
Wednesday"/>
    <Hour hourSet = "9AM"/>
  </StartTimeExpr>
  <DurationExpr cal = "Hours" len =
12/>
</PeriodicTimeExpr>
```

Figure 2.1 Periodicity Expression

Time: We use the periodic time expression represented by pairs of the form [I, P] and *calendars* to express timing constraint [Jos02]. A *calendar* is defined as a countable set of contiguous intervals. We write $C_a \sqsubseteq C_b$, 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 = $\sum_{i=1}^{n} O_i.C_i \triangleright x.C_d$, where C_d, C_1, …, C_n are calendars and $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$, $C_i \sqsubseteq C_{i-1}$ for i = 2,.., n, $C_d \sqsubseteq C_n$, and $x \in \mathbb{N}$. Expression on the left of $\triangleright$ identifies the set of starting points of the intervals, and that on the right side indicates the duration of each interval in terms of calendar C_d. For example, {*all.Years* + {3, 7}.*Months* $\triangleright$ 2.*Months*} represents the set of intervals that start on the third and seventh month of every year and have duration of 2 months [Jos02]. Figure 2.1 shows the XML specification for periodic time denoting every *Mondays* between 9am-9pm in year 2003. Calendars *Year*, *Month* and *Week* are by default assumed to be associated with *all* (e.g., *all.Year*).

Location: We use a session parameter to record the user domain associated with an access request to provide location-based access control. Additionally, the proposed language allows capturing attributes such as *login_time*, *login_date*, and *duration* of the session that profile user activities. Such information is processed dynamically and incorporated into the access decisions.

Content-based access: Content-based access control specification is allowed at four levels: *conceptual* level, *XML schema* level, *XML instance* level, and *XML element* level. 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 [Bha04]. Access control specification for the XML document sources can be done at the schema, instance or element level, as detailed later.

3. RBAC Policy Specification Framework for Multi-domain Environments

Multi-domain environments have manifested in various forms of emerging systems. Those particularly becoming prominent include *Web-services* and *Grid-based* systems [Azz02, Pea02]. *Web Services* are typically employed in

B2B applications where service providers expose specific information to clients, or an automated transaction is carried out between two e-commerce applications. *Grid-based* systems are emerging as a promising technology that can span an Internet size environment with heterogeneous systems distributed across multiple administrative domains [Azz02]. In a multi-domain environment, the key security goal is to ensure that no security violations occur during inter-domain accesses. In particular, secure interoperation should enforce the following two principles [Gon96]:

- The *autonomy principle*, which states that if an access is permitted within an individual system, it must also be permitted under secure interoperation.
- The *security principle*, which states that if an access is not permitted within an individual system, it must also not be permitted under secure interoperation.

Figures 3.1(*i*)(a)-(b) illustrate a violation of the security principle. Let *A, B, C, D, X, Y* and *Z* be roles and let the links indicate inheritance, i.e., 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 the links *a* and *b* indicate inter-domain accesses allowed between domains 1 and 2. Assuming the inter-domain links also have inheritance semantics, users authorized for role *C* are also authorized for roles *Y* and *Z*. It is easy to see that links *a* and *b* do not violate the above two principles in Figure 3.1(*i*)(*a*). Suppose, we add inter-domain links *c* and *d* as depicted in Figure 3.1(*i*)(*b*). This results in a violation of the security principle because 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 3.1(*ii*) depicts two architectural configurations that can characterize a multi-domain environment. We refer to them as *loosely coupled* and *tightly coupled* or *federated* multi-domain environments. Here, domains 3, 4 and 5 form a *federated* environment while domains 1 and 2 form a *loosely coupled* environment. Multi-domain environments may contain both *loosely coupled* and *federated* components. The arrows indicate general flow of access requests. The dotted lines indicate that access requests received by a domain for information in other domains are redirected to the global layer.

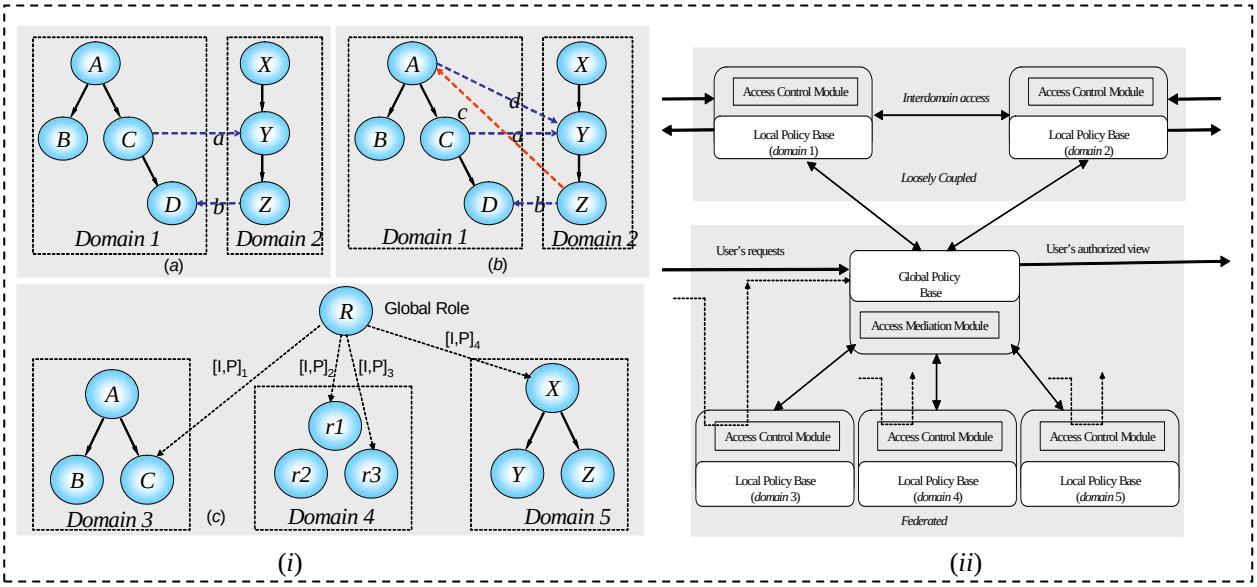

Figure 3.1 (*i*) An example multi-domain environment, (*ii*) Role Mappings in multi-domain environments

3.1 Loosely Coupled Multi-domain Environments

In a *loosely coupled* multi-domain environment, independent systems dynamically come together to share information for a period of time. An example of a *loosely coupled* environment is that of a company allowing its consulting firms to partially share information during the period of their contractual agreements. Two access mediation approaches can be employed in such a *loosely coupled* multi-domain environment:

1. Use a predefined set of role mappings to mediate inter-domain accesses. This approach requires the constituent systems to indicate the level of sharing they want to allow, and establish a consistent set of mediation rules for inter-domain accesses;

2. Use a certificate based approach to map unknown principals to predefined roles. This approach is suitable for an environment like the Internet where access is given to anyone presenting required credentials or role attribute values that satisfy role preconditions. Typically, this method relies on a trust management infrastructure [Bla99]

In either case, a trust negotiation phase may be involved.

3.2 Federated Multi-domain Environments

In a *federated* multi-domain environment, one system is typically designated as the master and others as local domains. The master is responsible for mediating accesses to individual systems by maintaining a global policy. For example, a digital government can be viewed as a *federated* system that attempts to provide users a set of services by federating a number of government units [Jos01]. Such systems may also characterize merged organizational systems, in which the policies are integrated. A grid can be considered a federated system where "donor" systems join it by submitting their local policies [Azz02, Pea02]. Typically, the global policy maps to local policies. For instance, a global role can map to various local roles in individual domains. Figure 3.1(*i*)(c) shows the mapping of a global role *R* to the local role *C* in *domain* 1, *r1* and *r2* in *domain* 2, and *X* in *domain* 3. Table 2 depicts such a mapping specific to a healthcare application environment.

Table 2. Example role mapping in a federated system

<table>
<tr><td>Mapped to→</td><td>Hospital 1 role C</td><td>Hospital 2 roles r1 and r2</td><td>Hospital 3 role X</td></tr>
<tr><td rowspan="2">R is FederatedDoctor</td><td rowspan="2">C = DayDoctor in PT_1 =$[I,P]_1$ representing every Mondays, and Wednesdays</td><td>r1 = DayDoctor in PT_2= $[I, P]_2$ representing every Tuesdays and Thursdays</td><td rowspan="2">X = SupervisorDoctor in PT_1 =$[I, P]_4$ representing every Weekends</td></tr>
<tr><td>r2 = EmergencyDoctor in PT_3 =$[I, P]_3$ representing every Fridays</td></tr>
</table>

Under this mapping, a doctor who needs to be cross-appointed to different hospitals at different times, for instance, can be assigned to the FederatedDoctor role between 9*am*-6*pm* on *Mondays* through *Saturdays*. This means during daytime between 9*am*-6*pm*, Dr. Smith can assume:

- DayDoctor role in *hospital* 1 on *Mondays* and *Wednesdays*, and in *hospital* 2 on *Tuesdays* and *Thursdays*,
- EmergencyDoctor role in *hospital* 2 on *Fridays*, and
- SupervisorDoctor role on *Saturdays* in *hospital* 3.

4. XML-based Specification Language for RBAC Model

We have developed an XML specification framework for expressing RBAC policies as well as mediation policies in both *loosely coupled* and the *federated* multi-domain environments, called X-RBAC.

Figure 4.1 shows the XML syntax for policy specification. The appendix to the paper gives details on the notation used to express the grammar of our XML language. Key policy component definitions include XML Role Sheet (XRS), XML User Sheet (XUS), XML Permissions Sheet (XPS) XML User-Role Assignment Sheet (XURAS), and XML Permission-Role Assignment Sheet (XPRAS); these correspond to various components of the RBAC model depicted in Figure 1.1(a). X-RBAC also allows specification of integrated policies by including other policy definitions as a component through <!-- Local Policy Definitions --> thus supporting the dynamic creation of articulated and complex multi-domain policies. Each constituent policy may be a local policy of a federated system or a policy of a partner domain in a loosely coupled environment. Local policy definitions are included or simply referred to by using *id*s of the local policies. If local policies are defined, then the set of relationships between the global policy and each of the local policies need to be defined through <!-- Policy Relationship Definitions-->.

```
<!-- 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 4.1 X-RBAC Policy specification format

4.1 X-RBAC Policy Specification

We now present an overview of each of the basic policy specification components.

XUS: An XUS is used to define users and their credentials types. Credentials are used when unknown users are to be provided access to the system. Credential values presented will be checked by the system to see if they satisfy assignment precondition for the requested role. Figure 4.2 shows the specification syntax. Credential type definition specifies the attribute list associated with a credential type. 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)})

Each attribute of a credential type may be defined as mand, to indicate that it is mandatory, or as opt, to indicate that it is optional. User definition may simply define user *name* and *user_id*, or additionally specify the assigned credentials that the user may carry. An XUS instance is shown in Figure 4.4(a). The `MaxRoles` tag indicates the maximum number of roles that a user can be assigned to.

```
<!-- XML User Sheet --> ::=
<XUS xus_id = ""]>
   [<!-- Definitions of Credential Types -->]
   <!-- User Definitions -->
</XUS>

<!-- Definitions of Credential Types --> ::=
<XCredType>
   [<!-- Credential Type Definition -->]+
</XCredType>

<!-- Credential Type Definition --> ::=
<CredType ct_id =( id)
          type_name = (type name)>
   <AttributeList>
      [<!-- Attribute Definition -->]+
   <AttributeList>
<CredType>

<!-- Attribute Definition --> ::=

<Attribute [attr_id =( id)]>
   <AttributeName
      usage = "mand | opt"
      type=(type)> (name)
   </AtriibuteName>
</Attribute>

<!-- User Definitions --> ::=
<Users>
   <User [user_id = (id)]>
      [<UserName> (name) </Userame> ]
      [<!-- User credential> ]
      <MaxRoles> (number) </MaxRoles>
   </User>
</Users>

<!-- User credential -->::=
<CredType [cred_type_id = (id)]
          [type_name = (type name)]>
   <!-- CredentialExpression -->
</CredType>
<!—Credential Expression -->::=
<AttributeValuePairs>
      [<(attribute name)> (attribute value)
   </(attribute name)>]+
</AttributeValuePairs>

<!-- XML Permission Sheet -->::=
<XPS>
   [<Permission [perm_id = (id)>
      <Object [id = (id)] [type = (ype)]>
         (name)     </Object>
      <Operation> (op) </Operation>
   </Permission>]+
</XPS>
```

Figure 4.2 Syntax for XUS / XPS

```
<!-- XML Role Sheet>::=
<XRS [xur_id = (id)]>
   <!-- Role Definitions >
   [<!-- Separation of Duty Definition>]
</XRS>

<!-- Role Definitions>::=
<Roles>
   [<!-- Role Definition>]+
</Roles>

<!-- Role Definition -->::=
<Role>
   <RoleName [r_id = (id]> (name)
   </RoleName >
   [<Attributes>
      [<AtributeName> (name)
      </AtributeName>]+
   </Attributes>]
   [<!-- Enabling Constraint --> ]
   [<!-- Activation Constraint --> ]
   [<Junior> (name) </Junior> ]
   [<Senior> (name) </Senior> ]
   [<Cardinality>(num)</Cardinality> ]
</Role>

<!—Assignment Condition-->::=
   [<AssignCondition [pt_expr_id =(id)]
                     [d_expr_id =(id)]>
      [<!—Logical Expression→]+
    </AssignCondition>]+
<!--Enabling Constraint --> ::=
<EnabConstraint [op=(AND|OR|NOT|XOR)]>
   [<EnabCondition [pt_expr_id =(id)]>
      [<!—Logical Expression→]+
   </EnabCondition>]+
<EnabConstraint>

<!-- Activation Constraint --> ::=
<ActivConstraint
[op=(AND|OR|NOT|XOR)]>
   [<ActivCondition [pt_expr_id =(id)]>
      [<!—Logical Expression→]+
   </ActivCondition>]+
<ActivConstraint>

<!—Logical Expression -->::=
<LogicalExpression
      [op = "AND | OR | NOT | XOR">
   [<!-- Predicate -->]+
</LogicalExpression>

<!-- Predicate -->:=
<Predicate>
   [<Operator> (lt|gt|eq|neq) </Operator>
   [<FuncName>(name)</FuncName>]
   [<ParamName>(name)</ParamName>]+
   <RetValue>(value)</RetValue>]
   |   <!—Logical Expression→]
</Predicate>
```

Figure 4.3 Syntax for XRS

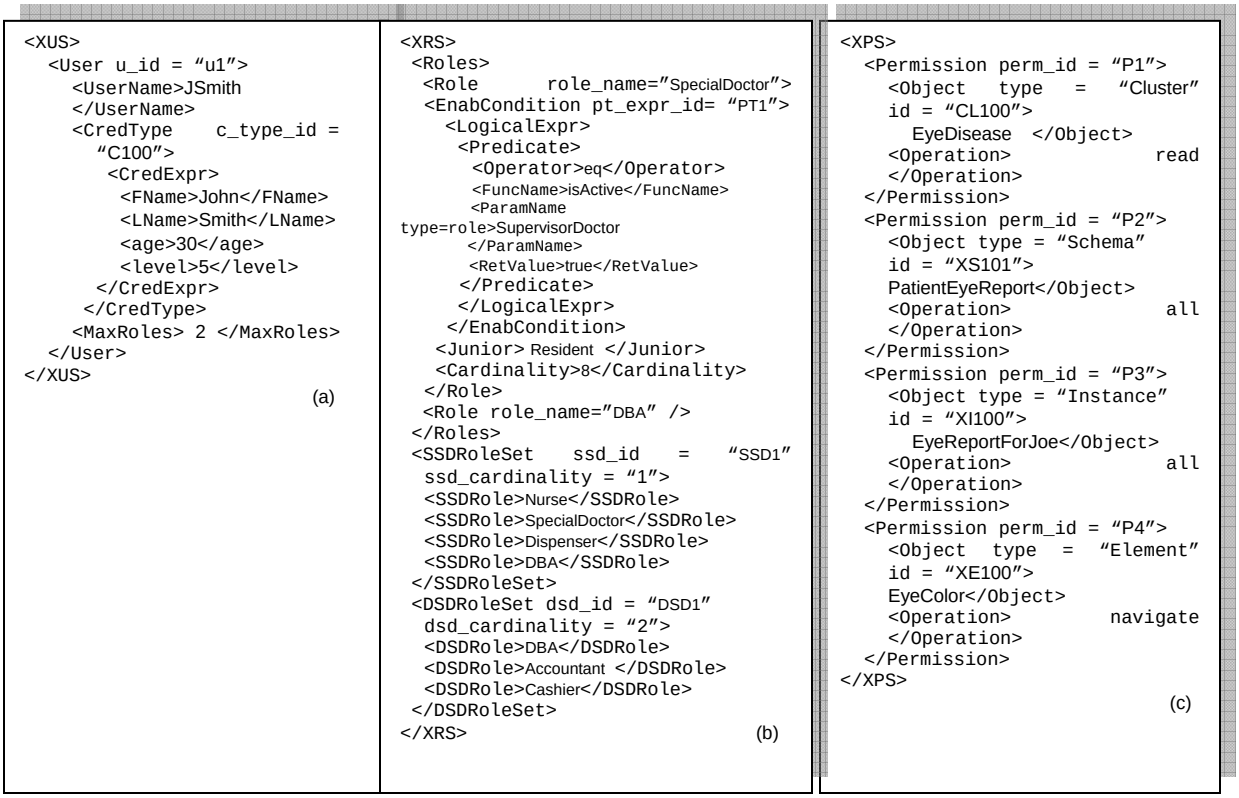

Figure 4.4 Examples of XUS, XRS and XPS

XPS: Permissions are specified in X-RBAC using the syntax for XPS shown in Figure 4.2. The permissions for a given system are defined in terms of *objects* and associated *operations*. Figure 4.4(c) shows an XML instance of permission specifications. Here, *perm_id* is a unique permission identifier. An *object* can represent either a (*i*) *cluster*, (*ii*) *schema*, (*iii*) *instance document*, or (*iv*) *document element*. We use *object types* to distinguish them. Clusters, schemas and documents are identified by their respective *ids* provided by the system administrator. *XPath* (XML Path Language) expressions are used to specify elements within an XML document. Having access privileges to a cluster implies that access to all schemas and instance documents belonging to the cluster is allowed Similar semantics apply to schema and its elements and instances. Permissions can have a propagation option that indicates whether or not it propagates down the object hierarchy [Ber01].

In Figure 4.4(c), the permission identified by "P1" allows a "*read*" operation on all documents within the scope of cluster identified by "CL100" with the default propagation option (i.e. "no prop"). Similarly, the permissions identified by "P2"and "P3" allow "all" operations on: (i) all document instances conforming to the schema identified by "XS101", and (ii) the document instance identified by "XI100", respectively, with the default propagation option. Lastly, the permission identified by "P4" allows "navigate" operation on the XML element "Name", also with the default propagation option.

XRS: Role definitions are provided in an XRS as shown in Figure 4.3. For each role, a set of role attributes is specified. Each role may have associated with it preconditions for its *enabling*, *assignment* and *activation*, that are separately defined using the <EnabCondition>, <AssignCondition> and <ActivCondition> tags. Within a precondition tag, an eType attribute may be specified to indicate whether the precondition is for the *enabling* (*activation*) or the *disabling* (*deactivation*). For enabling/disabling preconditions, we use the periodic time expression (Figure 2.1) as a condition. We may define additional predicates to be used to express context-based conditions using the generic syntax for logical conditions shown in Figure 4.3. Note that we may allow any complex logical expression using this syntax. A role definition may specify hierarchy relations by specifying its juniors and seniors using the <Junior> and <Senior> tags respectively , and express role cardinality using the

<Cardinality> tag. Separation of duty (SoD) constraints are specified by constructing a role set, and specifying a cardinality stating how many roles from the set may be assigned to (Static SoD), or activated (Dynamic SoD) by a user. An XML instance document describing SpecialDoctor and DBA roles along with the corresponding SSD (static SoD) and DSD (dynamic SoD) role sets is shown in Figure 4.4 (b). Accordingly, SpecialDoctor belongs to the SSDRoleSet identified by SSD1, with cardinality 1, and hence users may not be assigned to more than one role from this set. Similarly, the DBA role belongs to the DSDRoleSet identified by DSD1, with cardinality 2, and hence no more than two roles can be activated at the same time by authorized users.

XURAS, XPRAS: The system administrator uses XURAS and XPRAS to specify the user-role and permission-role assignment. The syntax is depicted in Figure 4.5. An XML instance of XURAS for assigning user credentials to a role is shown in Figure 4.6. This 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 "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.

```
<!--XML User-Role Assignment>::=
<XURAS>
   [<!-- User-Role Assignment>]+
</XURAS>
<!--User-Role Assignment>::=
<URA ura_id=(id)
     role_name (name)>
   <AssignUsers>
    <AssignUser [user_id = (id)]>
     [<!—Assignment Condition-->]
   </AssignUser>
</AssignUsers>
</URA>
<!--XMLPermission-Role
Assignment>::=
<XPRAS>
   [<!-- Permission-Role Assignment>]+
</XPRAS>
<!--Permission-Role Assignment>::=
<PRA pra_id=(id)
     role_name (name)>
   <AssignPermissions>
    <AssignPermission [perm_id =
(id)]>
     [<!—Assignment Condition-->]
    </AssignPermission>
   </AssignPermissions>
</PRA>
```

Figure 4.5. Syntax for XURAS

```
<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 4.6. An example of XURAS

```
<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 4.7. An example of XPRAS

Conceptual Level Specification: Conceptual level access control uses roles related to concepts. An instance of such a schema specification is shown in Figure 4.7. Here, the mapping identified by "PRM1" associates the EyeDoctor role with the permission "P1", which refers to an object cluster referring to a concept (see Figure 4.4(c)). In this case, an "EyeDoctor" role is authorized to "*read*" all the documents within the cluster identified by cluster id "CL100".

Schema, Instance and Element-level Specification: XML schemas, document instances and elements within can be protected similarly by associating them with corresponding roles. For instance, the mapping identified by "PRM2" in Figure 4.7 associates the DBA role with the permissions "P2" and "P3", which refer to a schema object and an instance document, respectively (see Figure 4.4(c)). In this case, the DBA role is authorized to "*read/write/navigate*" all instance documents conforming to the schema id "XS101", and the instance document "XI100". Similarly, the mapping identified by "PRM3" associates the Dispenser role with the permission "P4" referring to the Name element. Hence, the Dispenser role is authorized only to "*navigate*" the Name element in all conforming instance documents.

4.2 X-RBAC Specification of Mediation Policies

Within a policy definition, we can include local policy definitions using <!—Local Policy Definitions --> as shown in Figure 4.8. Note that each policy may itself be a global policy over a set of local domains. A relevant principle for mediation policies is the following scoping rule:

Scoping rule: *If a policy P becomes a local policy of a higher level policy, then P's local policy definitions and the policy relations are not known to the higher level policy.*

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

Figure 4.8. Definitions of local policies and mapping relations

```
<!-- 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 4.9. Syntax of policy relation

```
<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>
      <MappedTo>
    </RoleMapping>
  </InterDomainMapping>
</XPR>
```

Figure 4.10. Policy relation specification for example 3.1. (*i*)(c)

```
<!-- Policy Definition for Domains of Figure 3.1(i)(a) -->
<XPolicy policy_id = "Policy1">
  <!—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>
<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>
```

Figure 4.11. Policy relation specification for example 3.1.(*i*)(a)

The above rule indicates that, within a global policy definition, only the entities of its local policies and not those of constituent domains of these local policies are visible. This abstraction simplifies the meta-policy construction. However, if the higher level policy management must oversee the consistency of the overall federation, then this rule may need to be relaxed. With local policies included, we need to define the relationships among their policy entities with the global entities. The XML syntax for defining policy relationships is shown in Figure 4.9. Each global role may be mapped onto a number of local roles, which may belong to the same or different local domains. For each mapping, a condition may be specified. We require that the local roles onto which a global role can be mapped are included in the local policy definitions. Figure 4.10 illustrates the specification of the global to local role mapping defined in example 3.1(*i*)(c).

We can use the same structure to capture the mediation policies for the loosely coupled systems. In such a case, each of the local policy definition comprises a part of the domain policy of the partner domain. For example consider domains 1 and 2 of Figure 3.2(*i*), with policies *Policy 1* and *Policy 2*. In the specification of *Policy 1*, some entities of *Policy 2* are known, specifically roles *Y* and *Z* and appear in the <!—Local Policy Definitions --> section. Similarly, in *Policy 2*, a part of entity definitions of *Policy 1* (i.e., roles *C* and *D*) will appear as local policy definition. We note that the role-to-role mappings can be from one domain to the other, as shown in Figure 3.1(*i*)(a), or bidirectional. To capture mapping direction, we include the `<MappedFrom>...</MappedFrom>` syntax similar to the `<MappedTo>...</MappedTo>` syntax. Snapshots of relationship definitions for the role-to-role mapping of Figure 3.1(*ii*)(a) are shown in Figure 4.11.

5. Policy Integration Challenges in Multi-domain Environments

There are several issues such as semantic heterogeneity and policy consistency that pose considerable challenges in multi-domain environments. To manage semantic heterogeneity and integration of multiple heterogeneous policies in an XML-integrated multi-domain environment, approaches used for semantic integration of heterogeneous database schema may be useful. We have developed a policy integration methodology for a multi-domain environment, which consists of the following four phases:

- *Pre-integration* phase dealing with the bookkeeping aspects of integration involving semantic information about policy entities of each domain that facilitate resolving semantic differences. The overall process in this phase can use a data dictionary-based approach, or can involve building a common ontology.
- *Policy comparison* phase involving detection of semantic conflicts, including naming conflicts among domain roles, or structural conflicts among role hierarchies. This is facilitated by information obtained in the pre-integration phase. Techniques for detection of semantic conflicts, automatically or semi-automatically, need to be developed.
- *Policy conformance phase* that deals with the issue of resolving semantic and *rule* conflicts. Automatic techniques are needed to synthesize mediation policies when security violations such as one shown in Figure 3.1(i)(b) occur.
- *Merging and restructuring phase* that deals with needed readjustment to local policies for obtaining consistent merged policy after removing inconsistencies.

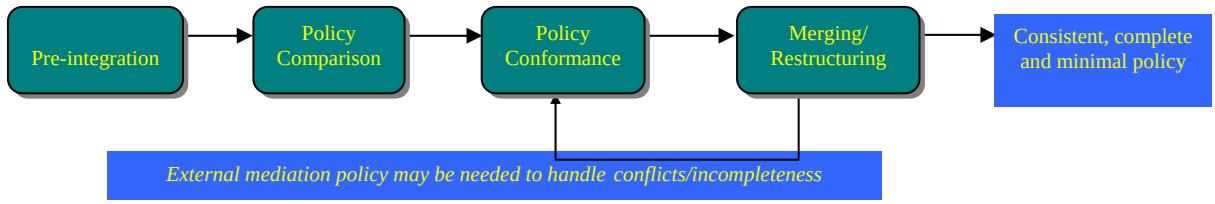

Figure 5.1 Policy Integration Phases

This process can be iterative in nature. In particular, the removal of inconsistencies can entail considerable restructuring and refinement of the mediation policies. Such iteration, in conjunction with the policy conformance phase, is carried out by the merging/restructuring phase.

5. Conclusion and Future Work

We have presented an XML-based policy specification language for expressing RBAC policies with temporal constraints. We have prototyped an X-RBAC system and demonstrated its use for Web-based access control

[Bha04]. We plan to extend our work in several directions. As the XUS maintains a lot of user data, a provision for user privacy is highly desirable. We plan to pursue X-RBAC extensions to provide a provision for privacy preferences. We also plan to extend our framework to allow interoperation of our mechanism with single sign-on mechanisms and apply it to securely compose interoperable web-services. Furthermore, we plan to extend the X-RBAC language to include the full set of temporal constraints introduced in the GTRBAC model [Jos02].

References

[Azz02] F. Azzedin, M. Maheswaran, "Towards Trust-Aware Resource Management", *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid* (CCGrid'02).

[Ber01] E. Bertino, S. Castano, E. Ferrari, "Securing XML Documents with Author X", *IEEE Internet Computing* May-June 2001.

[Bha04] Rafae Bhatti, James B. D. Joshi, Elisa Bertino, Arif Ghafoor, "XML-based Specification for Web-Services Document Security", *IEEE Computer*, Vol. 37, No. 4, April, 2004.

[Bla99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System Version 2," Internet RFC 2704, September 1999.

[Bon00] P. Bonatti, S. De Capitani di Vimercati, P. Samarati. A Modular Approach to Composing Access Control Policies. *Proc. of the Seventh ACM Conference on Computer and Communications Security*, Athens, Greece, November 1-4, 2000.

[Fer01] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, R. Chandramouli, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," *ACM Transactions on Information and System Security*, Vol 4, Issue 3, August 2001, pp. 224-274.

[Gon96] L. Gong and X. Qian, "Computational Issues in Secure Interoperation", *IEEE Transaction on Software and Engineering*, Vol. 22, No. 1, January 1996.

[Jos01] J. B. D. Joshi, A. Ghafoor, W. Aref, E. H. Spafford, "Digital Government Security Infrastructure Design Challenges", *IEEE Computer*, Vol. 34, No. 2, February 2001, pages 66-72.

[Jos02] J. B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, "Generalized Temporal Role Based Access Control Model", *Accepted for publication in IEEE Transaction on Knowledge and Data Engineering*.

[Ker02] A. Kern, "Advanced Features for Enterprise-Wide Role-Based Access Control", *Annual Computer Security Applications Conference*, 2002

[Ker03] Angelos D. Keromytis, Sotiris Ioannidis, Michael B. Greenwald, and Jonathan M. Smith, "The STRONGMAN Architecture," In Proceedings of the *3rd DARPA Information Survivability Conference and Exposition (DISCEX III)*, pp. 178 - 188. April 2003, Washington, DC.

[Osb00] S. L. Osborn, R. Sandhu, Q. Munawer, "Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM Transactions on Information and System Security*, Vol. 3, No. 2, February 2000, pp. 85-106.

[Pea02] L. Pearlman, V. Welch, Ian Foster, Carl Kesselman, S. Tuecke, "A Community Authorization Service for Group Collaboration," *2002 IEEE Workshop on Policies for Distributed Systems and Networks.*

[Pow00] R. Power, ""Tangled Web": Tales of Digital Crime from the Shadows of Cyberspace," *Que/Macmillan Publishing*, Aug. 31, 2000.

[URLa] XACML 1.0 Specification: http://xml.coverpages.org/ni2003-02-11-a.html

[URLb] XACML Profile for RBAC: http://xml.coverpages.org/OASIS-XACML-RBACProfile.pdf

[Vuo01] N. N. Vuong, G. S. Smith, Y. Deng, "Managing Security Policies in a Distributed Environment Using eXtensible Markup Language (XML)", *Symposium on Applied Computing*, March 2001.

Appendix

We follow the following notation to express compactly the grammar for our XML language. "<!-- X -->" indicates a term that needs to be defined; "[X]" indicates zero or one occurrence of term X. A + indicates repetition. "(value)" represents a constant value. "X:: *blockdef*" indicates that term X is expanded by definition *blockdef*. `<Xyz [expr1] [expr2] expr3>` represents actual XML tag `Xyz` and optional expressions `expr1` and `expr2` and mandatory expression `expr3`.