A Policy Validation Framework for Enterprise Authorization Specification

Ramaswamy Chandramouli
National Institute of Standards and Technology, Gaithersburg, MD, USA
(mouli@nist.gov)

Abstract

The validation of enterprise authorization specification for conformance to enterprise security policies requires an out-of-band framework in many situations since the enforcing access control mechanism does not provide this feature. In this paper we describe one such framework. The framework uses XML to encode the enterprise authorization specification, XML Schema to specify the underlying access control model (which in our case is the Role-based Access control Model (RBAC)) and Schematron language to encode the policy constraints. The conformance of the XML-encoded enterprise authorization specification to the structure of the RBAC model (specified through XML Schema) as well as the policy constaints (specified through Schematron) are verified through a Schematron Validator tool.

1. Introduction

An access control mechanism provided by or within any software (e.g. operating system, DBMS etc) is the executable module for controlling access to resources under the control of the software. Every access control mechanism provides a structural framework called the access control model for specifying access restrictions for resources. An access control model is based on certain concepts involved in interaction with resources. These concepts can be broadly described as being made up of entities (e.g., subject, object, operations, permission or right, user, role, label, group etc) and relations (the combination of an object and operation defines a permission) that describe the nature of association between entities. The deployment of an access control model for an enterprise environment is called a configuration. The configuration of an access control model for a given enterprise contains instances of model entities for that enterprise (e.g., role instances like Teller, Loan Officer for a commercial bank environment) and hence results in the Enterprise Authorization Specification. The safety of a configuration is defined as the state where the configuration does not violate enterprise access control/authorization policies. To verify the safety of the configuration therefore requires that the policies be expressed using access control model entities and relations. In other words there should not be any policy violations by the instances of the model entities and model relations for the enterprise. A common approach adopted to meet this requirement is to augment the access control model with expressions called constraints.

There are however many practical limitations in ensuring that the enterprise authorization specification is safe (does not violate policy constraints). The first one is the limitation of the underlying access control model. Since policies are specified using model entities and relations, it should be obvious to many practitioners that some access control models are more amenable for expression of complex enterprise policies than others. In general, higher the level of abstraction of model entities more is the policy definition capabilities of the model. Secondly, even if the underlying access control model does provide policy definition capabilities, the access control mechanism may not provide features for specification of all the different types of constraints needed to capture those policy requirements. The above two limitations point the need for an out-of-band approach (independent of access control mechanism and the underlying software platform) to represent enterprise authorization specification and validate it for satisfaction of enterprise policy constraints.

In this paper we provide one such approach. We have represented the enterprise authorization specification for a commercial bank enterprise in XML. The authorization specification is based on the role-based access control model (RBAC) [1]. The RBAC model itself is specified using XML Schema [4]. The RBAC XML Schema specification is then augmented with policy constraints using the Schematron constraint specification language [9]. The XML document containing the bank-enterprise authorization specification is then validated using the Schematron Validation Tool [12].

The organization of the rest of the paper is as follows. In Section 2 we provide an overview of the various

components in our policy validation framework as well as the rationale for their choice. Section 3 describes the specification of the RBAC model using XML Schema language. Since RBAC model standards [10] provide taxonomy of models (as opposed to a single RBAC model) as well as well as choice of features to suit the enterprise environment, we will call our customized RBAC model for our bank environment as Bank-RBAC model. We will refer to the specification of Bank-RBAC model in XML Schema as Bank-RBAC XML Schema and the XML encoding of the bank-enterprise authorization specification based on the RBAC XML Schema as Bank-Authorization XML Data. Section 4 provides a sample encoding in XML of bank-enterprise authorization specification. Section 5 deals with constraints that could be specified using the XML-Schema language features. These cover all Bank-RBAC model's model-specific constraints as well as some rudimentary application-domain constraints. In section 6 we point out the limitations of the XML Schema features for specification of complex policy (domain-specific) constraints. We then illustrate the use of Schematron language for expressing those domain constraints within our Bank-RBAC XML Schema document with several examples. We briefly describe related work in section 7 and in section 8 we explain the scope for extending our current framework to enhance the capabilities of access control mechanism to enforce dynamic policy constraints.

2. Policy Validation Framework Components

A framework for programmatic or tool-based validation of enterprise authorization specification should have the following components:

(a) Choice of the underlying access control model and a language for its specification
(b) A language for encoding enterprise authorization specification based on the access control model
(c) A language for specifying policy requirements as constraints based on the access control model
(d) A tool or API for programmatic validation of the enterprise authorization specification for conformance to model specifications and policy constraints.

2.1 Choice of Access Control model and its specification

Our motivation for choosing RBAC as the underlying access control model for the bank-enterprise authorization specification is that it is a sufficiently abstract model with configurations capable of expressing varied types of policies such as least privilege and separation of duties. RBAC has been widely implemented for different types of products such as Database Management Systems, Workflow systems and Enterprise Security Management systems [18]. A brief description of RBAC Models is as follows.

The Role-based Access Control Model (RBAC) provides a generalized approach for representation of many types of access control policies (each describable only using a specific access control model) through the abstraction concept of roles. Many RBAC models have been proposed in the research literature [2] and the NIST RBAC standard provides taxonomy of RBAC models [10]. The RBAC reference model in the standard has four main entities – users, roles, privileges and sessions. Roles generally represent organizational functions (e.g. Teller in a bank). Users are assigned to roles and privileges are assigned to roles as well. Users derive all their privileges by virtue of their role memberships. Users interact with the system through sessions and roles are assigned to particular sessions as well. Now the interactions among these four entities of the RBAC model results in the following relations:

(a) Role-Inheritance relation (RH)
(b) User-Role relation (UA)
(c) Privilege-Role relation (PA)
(d) User-Session relation (US)
(e) Role-Session relation (RS).

A schematic diagram of our reference RBAC model is given in Fig 1.

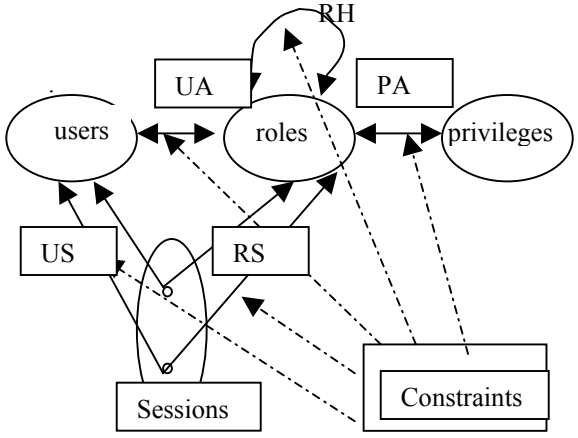

Figure 1

The Bank-RBAC model that we have chosen for illustration in our policy validation framework is based on the RBAC reference model described above but without the session entity and its two associated relations US and RS. We have excluded the session entity since session is a platform-dependent artifact. For example, a DBMS

session has a different set of parameters than a O/S login session like Telnet. We have also excluded the constraint from the model specification per se since constraint needs a different set of enforcement mechanisms than the simple set of cardinality and participation restrictions associated with binary relations. To summarize our Bank-RBAC model consists of users, roles and privileges as entities and the following relations – Role Inheritance (RH), user-role relation (UA) and privilege-role relation (PA).

Our language for Bank-RBAC model specification is XML Schema [4] since it provides constructs for specifying binary relations and hierarchical structures (the basic structural relationships of the RBAC model). XML Schema is one of the languages under the XML standard that is used for describing the structure of information within an XML document. Our choice of XML Schema over the other meta-data language DTD is due to the fact that XML Schema supports specification of cardinality and participation restrictions as well as rich data types (like enumerated data types). Further we need a means to augment the specification of the Bank-RBAC model with policy constraints. The XML Schema language enables this feature as well by allowing the embedding of constraints in other languages under a special "annotation" tag. We have made use of this feature by embedding our policy constraints specified using the Schematron language [9] within the XML-Schema representation of our Bank-enterprise RBAC model.

2.2 A Language for Encoding Enterprise Authorization Specifications

Our choice of XML Schema for Bank-RBAC model automatically provides XML as the choice for encoding enterprise authorization information. An advantage of encoding a structured information (such as bank-enterprise authorization specification) in XML is that there are special types of software called XML Parsers that could be used to extract information from XML documents based on its associated structure (that is specified through XML Schema document). These XML Parsers are based on standard application programming interfaces such as Document Object Model (DOM) [5]. These parser libraries implemented in various procedural languages enable an application program written in the corresponding procedural language to create, maintain and retrieve XML encoded data. With an API for extracting information, a program could be written to properly interpret the contents of the validated enterprise authorization specification (encoded in XML), and map them to the native access control structures in the access control mechanisms present in heterogeneous application systems within the enterprise.

It is useful to point out at this stage that XML Parsers can also be used to validate an XML document for conformance to the structure specified in an associated XML Schema document. Hence in our case the Bank-Authorization XML Data document can be validated for conformance to Bank-RBAC model specified through a XML Schema document. However as we pointed out earlier, XML Schemas can only be used for specifying data typing and cardinality constraints. These constraints are useful for properly specifying model entities and their associated binary relations. Hence XML Schemas can specify model-based constraints and therefore can be used to validate whether the Bank-Authorization XML data does indeed conform to the particular adaptation of the RBAC model for our banking enterprise (i.e., Bank-RBAC model).

2.3 A Language for specifying policy constraints

We already alluded to the fact that the XML Schema with its support for data types, cardinality and participation constraints can handle structural constraints and hence all model-based constraints (being structural in nature) can be expressed through XML Schema. However policy constraints pertain to the enterprise domain and hence involve the contents of enterprise authorization specification. More specifically they involve the model entity and relation instances found in the Bank-Authorization XML Data. Further, studies have shown [6,7] that the content-based policy constraints are much more complicated than model-based constraints since they may involve complex logical expressions or rules.

One approach that has been adopted to represent domain constraints is to annotate an XML Schema that has been used for representing a model for a domain, with ontological information regarding the domain using pattern based languages such as RDF [8] and Schematron [9]. In this paper we have annotated the XML Schema for Bank-RBAC Model with Schematron constraints that specify rules that that the access control data (in Bank-Authorization XML Data) pertaining to the bank enterprise domain has to satisfy.

2.4 A Tool or API for Validation of Enterprise Authorization Specification

We have used a tool called the Schematron Validator for validating the bank-enterprise authorization specification (in Bank-Authorization XML Data) for conformance to policy constraints specified through the Schematron language. Since the Schematron Validator tool also validates an XML document for conformance to the referenced structure, it also automatically checks the XML encoded bank-enterprise authorization specification for conformance to the Bank-RBAC model specified through XML Schema. Hence using this tool we can validate the Bank-Authorization XML Data for

satisfaction of both model-based as well as content-based (policy) constraints.

3. XML Schema specification of Bank-RBAC Model

The basic artifact for modeling any concept in XML Schema is the element. A name, type and a set of attributes can be specified for an XML Schema element. The type can be a simple data type like a 'string', or a complex data type. A complex data type in turn may involve additional elements. A data type can be an enumerated type (can only assume a value from a given set) as well. In addition a special data type called 'ID' is supported. This is often used as the data type for an attribute if that attribute uniquely identifies an instance of that element.

It is possible to specify certain structural constraints associated with an element. We can specify the maximum and minimum of times that element instance can occur in the XML document based on the XML Schema specification. We can also specify whether the use of an element or attribute is mandatory or optional.

As far as our Bank-RBAC model is concerned, all the entities (User, Role, Privileges) as well as relations (User-Role relation (UA), Role-Inheritance relation (RH) and Privilege-Role relation (PA)) are modeled as elements. Since these entities either contain multiple attributes (as in the case of elements representing User, Role and Privileges) or sub elements (as in the case of UA, RH and PA) relations, the data type associated is always a complex data type.

The specification of the User entity is as follows:

```
<xs:element name="user"     type="userType"/>
<xs:complexType name="userType">
<xs:attribute name="userID" type="xs:ID"
              use="required"/>
<xs:attribute name="fullname" type="xs:string"
              use="optional"/>
</xs:complexType >
```

The above definition of the data type 'userType' means that a user is represented as having two attributes 'userID' and 'fullname' with the former declared as a mandatory attribute and the latter declared as an optional attribute. Please note that the data type for 'userID' attribute is designated as 'xs:ID' which implies that the value for 'userID' attribute must be unique and hence no duplicates are allowed.

The entity 'Role' is specified as follows:

```
<xs:element name="role"   type="roleType"/>
<xs:complexType name="roleType">
<xs:attribute name="roleID" type="xs:ID" use="required"/>
<xs:attribute name="rolename" type="validRole"
              use="required"/>
<xs:attribute name="cardinality" type="roleLimit"
              use="optional"/>
</xs:complexType>
```

To complete our definition of role component, we need to define the data types "validRole" and "roleLimit". The data type definition of "validRole" lists the set of permissible role names in the bank enterprise while that for the "roleLimit" is used to specify a number that stands for the minimum and maximum number of users that can be assigned to that role.

```
<xs:simpleType name="validRole">
    <xs:restriction base="xs:string">
<xs:enumeration value="BranchManager"/>
<xs:enumeration value="Customer_Service_Rep"/>
<xs:enumeration value="SD_Vault_Officer"/>
<xs:enumeration value="Loan_Officer"/>
<xs:enumeration value="Accounting_Manager"/>
<xs:enumeration value="Internal_Auditor"/>
<xs:enumeration value="Teller"/>
<xs:enumeration value="Accountant"/>
  </xs:restriction>

</xs:simpleType>
<xs:simpleType name="roleLimit">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
    <xs:maxInclusive value="10"/>
   </xs:restriction>
</xs:simpleType>
```

The privilege is a combination of a resource and operation. The privilege entity of the Bank-RBAC model is specified as:

```
<xs:element name="privilege" type="privilegeType"/>
<xs:complexType name="privilegeType">
<xs:attribute name="privID" type="xs:ID" use="required"/>
<xs:attribute name="resource" type="xs:string"
              use="required"/>
<xs:attribute name="oper" type="operType"
              use="required"/>
</xs:complexType>

<xs:simpleType name="operType">
    <xs:restriction base="xs:string">
<xs:enumeration value="Open"/>
<xs:enumeration value="Close"/>
<xs:enumeration value="Debit"/>
<xs:enumeration value="Credit"/>
  </xs:restriction>
</xs:simpleType>
```

We now provide the XML Schema representation for the User-Role Assignment (UA) relation of the Bank-RBAC model.

```
<xs:element name="UserRoleAssignment"
            type="URAType"/>
<xs:complexType name="URAType">
    <xs:sequence>
        <xs:element name="user" type="xs:IDREF"
                    maxOccurs="10"/>
    </xs:sequence>
    <xs:attribute name="role" type="xs:IDREF"
             use="required"/>
</xs:complexType>
```

The XML Schema representation for the role-inheritance relation (RH) is as follows:

```
<xs:element name="role_inherit" type="InheritType"/>
<xs:complexType name="InheritType">
    <xs:attribute name="Inherit_ID" type="xs:ID"
            use="required"/>
   <xs:attribute name="FromRole" type="validRole"
            use="required"/>
   <xs:attribute name="ToRole" type="validRole"
            use="required"/>
</xs:complexType>
```

The Privilege-Role relation (PA) is specified in XML Schema as:

```
<xs:element name="RolePrivilegeAssignment"
type="RPAType"/>
<xs:complexType name="RPAType">
  <xs:sequence>
        <xs:element name="privilege" type="xs:IDREF"
maxOccurs="unbounded"/>
  </xs:sequence>
<xs:attribute name="role" type="xs:IDREF"
use="required"/>
</xs:complexType>
```

The specification of a construct to specify conflicting roles for checking static separation of duty constraints is specified as:

```
<xs:element name="ssd_roles" type="SSDType"/>
<xs:complexType name="SSDType">
<xs:attribute name="SSD_ID" type="xs:ID"
            use="required"/>
<xs:attribute name="BaseRole" type="validRole"
            use="required"/>
<xs:attribute name="ConflictRole" type="validRole"
            use="required"/>
</xs:complexType>
```

Finally the fact that the entire Bank-RBAC model is made of entities User, Role, Privilege and UA, RH and PA relations is specified in the XML Schema by creating a root element called 'BANK_RBAC_Model' with elements representing the entities and relations as sub-elements.

```
<xs:element name="Bank_RBAC_Model"
            type="BankRBACModelType"/>
<xs: complexType name="BankRBACModelType"
<xs:sequence>
    <xs:element ref="user" maxOccurs="unbounded"/>
    <xs:element ref="role" maxOccurs="unbounded"/>
    <xs:element ref="privilege"
            maxOccurs="unbounded"/>
    <xs:element ref="role_inherit"
            maxOccurs="unbounded"/>
    <xs:element ref="ssd_roles"
            maxOccurs="unbounded"/>
    <xs:element ref="UserRoleAssignment"
            maxOccurs="unbounded"/>
    <xs:element ref="RolePrivilegeAssignment"
            maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

Observe that some of the elements specified above do not have the name attribute (like other element definitions we have seen before) but refers to the already defined elements through the value specified in the 'ref' attribute. The above XML Schema definition was verified to be syntactically correct using the XML Schema Validator tool – XML Spy [11].

4. Encoding the enterprise authorization specification in XML

Now that we have developed an XML Schema specification of the Bank-RBAC model, we now encode the enterprise authorization specification in an XML document whose tag structure should correspond to the element definitions in the XML Schema.
We represent a sample set of users (by providing instances of the 'user' element in XML schema) as given below:

```
<user userID="DrayJ" fullname="Jim Dray"/>
<user userID="GranceT" fullname="Tim Grance"/>
<user userID="VincentH" fullname="Vincent Hu"/>
```

A sample set of encodings for role instances is:

```
<role roleID="BRM" rolename="BranchManager"
                    cardinality="1"/>
<role roleID="CSR" rolename="Customer_Service_Rep"
                    cardinality="3"/>
<role roleID="SDV" rolename="SD_Vault_Officer"
                    cardinality="2"/>
```

A sample set of privileges are given below:

```
<privilege privID="OPEN_ACCT" resource="DepAcct"
          oper="Open"/>
<privilege privID="DEBIT_ACCT" resource="DepAcct"
          oper="Debit"/>
```

```
<privilege privID="CREDIT_ACCT" resource="DepAcct"
          oper="Credit"/>
<privilege privID="CLOSE_ACCT" resource="DepAcct"
          oper="Close"/>
<privilege privID="APPROVE_LOAN"
          resource="LoanForm" oper="Update"/>
```

A sample set of User-Role relations is:

```
<UserRoleAssignment role='BRM'>
          <user>GranceT</user>
          <user>JansenW</user>
</UserRoleAssignment>
<UserRoleAssignment role=''ACC'>
          <user>VincentH</user>
</UserRoleAssignment>
<UserRoleAssignment role='LNO'>
          <user>TomK</user>
          <user>JohnW</user>
          <user>SusanW</user>
</UserRoleAssignment>
```

A sample of set Role-Inheritance relations is:

```
<role_inherit  Inherit_ID="HY1" FromRole="Teller"
ToRole="Customer_Service_Rep"/>
<role_inherit  Inherit_ID="HY2"  FromRole="Accountant"
ToRole="Accounting_Manager"/>
<role_inherit  Inherit_ID="HY6"
FromRole="Internal_Auditor"
 ToRole="BranchManager"/>
```

A sample set of encodings for Role-Privilege relations is:

```
<RolePrivilegeAssignment role='CSR'>
        <privilege>OPEN_ACCT</privilege>
       <privilege>CLOSE_ACCT</privilege>
       <privilege>DEBIT_ACCT</privilege>
      <privilege>CREDIT_ACCT</privilege>
</RolePrivilegeAssignment>
<RolePrivilegeAssignment role='LNO'>
       <privilege>CSR</privilege>
       <privilege>APPROVE_LOAN</privilege>
       <privilege>DEBIT_LOAN</privilege>
       <privilege>CREDIT_LOAN</privilege>
</RolePrivilegeAssignment>
```

A sample set of encoding for specifying conflicting roles is:

```
<ssd_roles  SSD_ID="SSD1" BaseRole="Internal_Auditor"
                              ConflictRole="Accountant"/>
<ssd_roles  SSD_ID="SSD2" BaseRole="Internal_Auditor"
                    ConflictRole="Accounting_Manager"/>
<ssd_roles  SSD_ID="SSD3" BaseRole="Internal_Auditor"
                        ConflictRole="BranchManager"/>
```

5. Constraints expressed through XML Schema

Let us now review the structural constraints (model-based) and some rudimentary domain-specific policy constraints that we have been able to specify in our Bank-RBAC XML Schema. (Please note that here we include only constraints that can be validated by an XML Schema parser).

5.1 Structural Constraints (Model-based) represented using the XML Schema

(a) Specification of mandatory and optional attributes - specifies what attributes should be specified while defining instances of Bank-RBAC model entities in the bank-enterprise authorization specification.

(b) Identification of attribute whose values must be unique (no duplicates allowed) – specifies the attribute that should have a unique value among all instances of a particular model entity (e.g. Role), or rather the identifier for an entity instance.

(c) Cardinality constraints showing the number of times (instances) an Bank-RBAC model entity can occur in the bank-enterprise authorization specification.(e.g. no more than (say) 10 roles can be there in the bank enterprise).

5.2 Domain Constraints represented using the XML Schema

(a) The role names that occur in Bank-Authorization XML Data should be one of the valid names from the list specified in the XML Schema (through the validRole data type).

6. Specification of Domain-specific Policy Constraints

We have seen that the only domain-specific policy constraint that XML Schema can specify is the list of valid names for Bank-RBAC model entity instances through the enumeration data type. No other content-based policy constraint can be specified using XML Schema. For specifying these types of constraints we use the Schematron language. In this section we illustrate with several examples the specification of policy constraints using Schematron.

In a schematron constraint definition, constraints are defined using the following tags:

(a) a ‘rule’ tag to define the context (in terms of the XML schema element) for the constraint and

(b) one or more ‘assert’ tags: Each ‘assert’ tag contains the Boolean expression for the property that each of the instances of the element (named in the context) has to

satisfy. Any violation of the property will be flagged off as an error.
(c)one or more 'report' tags: Each 'report' tag contains the Boolean expression for the property that each of the instances of the element (named in the context) should not satisfy. Any instance where the property is satisfied will be flagged off as an error.
(d)A set of 'diagnostic' tags: Each of these provides information on the violating data.
(e) The above tags are enclosed within a named 'pattern' tag.

With the above primer on Schematron, we now illustrate the specification of some important policy constraints that govern the access control requirements for the bank enterprise environment.

Constraint 1: (Role Cardinality Constraint) The cardinality limit (the maximum number of users that can be assigned) specified in the role definition for a role should not be violated in the actual user assignments for that role.

The role definition for the Branch Manager role(roleID = 'BRM') in our XML encoded access control data file is as follows:

```
<role roleID="BRM" rolename="BranchManager" cardinality="1"/>
```

The reference to the above data through the XML Schema components forms the context. The context therefore is a role instance definition whose roleID attribute is 'BRM" (for Branch Manager). This context is expressed in schematron as:

```
<sch:rule
context="Bank_RBAC_Model/role[@roleID='BRM']">
```

The assertion to be made in this context is that in the corresponding User-Role relation (where the @role='BRM'), the count of the number of users should not exceed the number specified through the cardinality attribute (@cardinality = 1). The assertion and the corresponding diagnostic messages expressed in schematron through the assert and diagnostic tags respectively are given below:

```
<sch:pattern name="Checking for Role Cardinality">
  <sch:rule
context="Bank_RBAC_Model/role[@roleID='BRM']">
    <sch:assert test = "./@cardinality >=
count(../UserRoleAssignment/user[../@role = 'BRM']) "
diagnostics="Cardinality_Exceeded">Cardinality for the
role exceeded </sch:assert>
    <sch:diagnostics>
      <sch:diagnostic id="Cardinality_Exceeded">The
actual number of users assigned is: <sch:value-of
<sch:assert test = "./@cardinality >=
   count(../UserRoleAssignment/user[../@role = 'BRM'])
  "diagnostics="Cardinality_Exceeded">Cardinality for
    the role exceeded
 </sch:assert>
<sch:diagnostics>
   <sch:diagnostic id="Cardinality_Exceeded">The
     actual number of users assigned is:  <sch:value-of
     s elect="count(../UserRoleAssignment/user[../@role
    = 'BRM'])"/> while cardinality limit is: <sch:value-
    of  select="./@cardinality"/>
  </sch:diagnostic
</sch:diagnostics>
</sch:rule>
</sch:pattern>
```

The actual data in our Bank-Authorization XML Data file is:

```
<UserRoleAssignment role='BRM'>
          <user>GranceT</user>
          <user>JansenW</user>
</UserRoleAssignment>
```

The schematron validator therefore generated the following error message:

```
From pattern "Checking for Role Cardinality":
   Assertion fails: "Cardinality for the role exceeded"  at
      /Bank_RBAC_Model[1]/role[1]
      <role roleID="BRM" rolename="BranchManager"
cardinality="1">...</> The actual number of users assigned
is: 2 while cardinality limit is: 1
```

Constraint 2: (Inheritance Integrity Constraint): Two conflicting roles (specified in the Static Separation of Duty specification) cannot inherit each other. For example the constraint that the role that conflicts with the Internal Auditor role cannot inherit that role is specified as:

```
<sch:pattern name="Checking for Inheritance Integrity">
 <sch:rule
context="Bank_RBAC_Model/role_inherit[@FromRole
='Internal_Auditor']">
   <sch:assert test= "not(@ToRole =
(../ssd_roles[@BaseRole='Internal_Auditor']/@ConflictRol
e))" diagnostics="INT_INTEG">A conflicting role cannot be
inherited.
    </sch:assert>
  <sch:diagnostics>
      <sch:diagnostic id="INT_INTEG">The violating
inheritance assignment is made for the role: <sch:value-of
select="./@ToRole"/>
        </sch:diagnostic>
      </sch:diagnostics>
  </sch:rule>
</sch:pattern>
```

The role that violates this inheritance integrity constraint in our bank-enterprise authorization specification is identified and the following diagnostic message is generated by the Schematron Validator tool:

```
From pattern "Checking for Inheritance Integrity":
   Assertion fails: "A conflicting role cannot be inherited."
at       /Bank_RBAC_Model[1]/role_inherit[6]
         <role_inherit Inherit_ID="HY6"
```

FromRole="Internal_Auditor" ToRole="BranchManager">...</> The violating inheritance assignment is made for the role: BranchManager

Constraint 3: (Static Separation of Duty Constraint): A user assigned to the Internal Auditor role (@role='AUD') should not be assigned to the Accountant role (@role='ACC') since Internal Auditor and Accountant are conflicting roles.

The context, the assertion and the diagnostic tags used to specify the above constraint is as follows:

```
<sch:pattern name="Checking for Separation of Duty">
<sch:rule
context="Bank_RBAC_Model/UserRoleAssignment[@role
='AUD']/user">
 <sch:assert test= "not(text() =
(../../UserRoleAssignment[@role='ACC']/user/text() ))"
diagnostics="SOD_AUD">There should not be a common
user in Audit and Accounting roles.
 </sch:assert>
 <sch:diagnostics>
      <sch:diagnostic id="SOD_AUD">The SOD violating
assignment is made for user: <sch:value-of
select="text()"/>
      </sch:diagnostic>
  </sch:diagnostics>
  </sch:rule>
```

For our Bank-Authorization XML Data, the schematron validator generated the following message:

From pattern "Checking for Separation of Duty":
Assertion fails: "There should not be a common user in Audit and Accounting roles." at /Bank_RBAC_Model[1]/UserRoleAssignment[6]/user[1]
<user>...</> The SOD violating assignment is made for user: VincentH

Constraint 4: (Constraint specifying Conflicting Users): Users John Wack (user/text() ='JohnW') and Susan Wack (user/text() = 'SusanW') should not be assigned to the same role (whatever be the role) since they have spousal relationship.

The schematron description of the above constraint is:

```
<sch:pattern name="Checking for Conflicting Users">
  <sch:rule
    context="Bank_RBAC_Model/UserRoleAssignment">
    <sch:assert test="2 > count (user [text () = 'JohnW'])
                        + count(user [text() = 'SusanW'])"
          diagnostics="Wack_Violate">John Wack and
      Susan Wack should not be assigned to the same
role
     </sch:assert>
    <sch:diagnostics>
      <sch:diagnostic id="Wack_Violate">The violating
            assignment is for the role: <sch:value-of
            select="@role"/>
       </sch:diagnostic>
      </sch:diagnostics>
   </sch:rule>
 </sch:pattern>
```

The diagnostic message prints out the role that JohnW and SusanW are assigned:

From pattern "Checking for Conflicting Users":
Assertion fails: "John Wack and Susan Wack should not be assigned to the same role" at /Bank_RBAC_Model[1]/UserRoleAssignment[4]
<UserRoleAssignment role="LNO">...</> The violating assignment is for the role: LNO

Constraint 5: (Constraint specifying dependent role assignments): Every user assigned to Safe Deposit Vault role (@role='SDV') should already be assigned to Customer Service Representative role (@role='CSD'). The Schematron syntax for the above constraint is:

```
<sch:pattern name="Checking for Dependent Role
                    Assignments">
   <sch:rule
context="Bank_RBAC_Model/UserRoleAssignment[@role
='SDV']/user">
     <sch:assert test="text() =
(../../UserRoleAssignment[@role='CSR']/user/text())"
diagnostics="SDV_CSR_Depend">A user assigned to
SDV must already be assigned to CSR role
    </sch:assert>
    <sch:diagnostics>
      <sch:diagnostic id="SDV_CSR_Depend">The
following user is assigned to SDV role but not to CSR role:
<sch:value-of select="text()"/>
     </sch:diagnostic>
    </sch:diagnostics>
   </sch:rule>
 </sch:pattern>
```

The diagnostic message due to our authorization specification not conforming to the above constraint is:

From pattern "Checking for Dependent Role Assignments":
Assertion fails: "A user assigned to SDV must already be assigned to CSR role" at /Bank_RBAC_Model[1]/UserRoleAssignment[3]/user[2]
<user>...</> The following user is assigned to SDV role but not to CSR role: Gray

Constraint 6: (Limits on Role Assignment for a specific user): The specification of the constraint that a particular user Tom (user/text()= 'TomK') should not be assigned more than two roles is:.

```
<sch:pattern name="Checking for limit on Tom's
Assignments">
  <sch:rule context="Bank_RBAC_Model">
    <sch:assert test="3 >
count(UserRoleAssignment[user/text()='TomK'])"
diagnostics="Tom_Limit">Tom should be assigned a
maximum of 2 roles
      </sch:assert>
```

```
    <sch:diagnostics>
      <sch:diagnostic id="Tom_Limit">The actual number
of roles assigned to Tom is: <sch:value-of
select="count(UserRoleAssignment[user/text()='TomK'])"/>
        </sch:diagnostic>
    </sch:diagnostics>
  </sch:rule
 </sch:pattern>
```

The diagnostic message generated on our authorization specification is:

```
From pattern "Checking for limit on Tom's Assignments":
    Assertion fails: "Tom should be assigned a maximum of
2 roles"  at
       /Bank_RBAC_Model[1]
      <Bank_RBAC_Model
xsi:noNamespaceSchemaLocation="A:\BankRBAC.xsd">..
.</> The actual number of roles assigned to Tom is: 3
```

Constraint 7: (Least Privilege Constraint): The right to open an account as well as to close an account should not be assigned to the same role. The Schematron constraint can be specified not only to verify whether a role with both privileges exists in the Bank-Authorization XML Data but also to print out the violating Role. The constraint specification is:

```
<sch:pattern name="Excess Privilege for a Role">
  <sch:rule
context="Bank_RBAC_Model/RolePrivilegeAssignment">
    <sch:assert test="2 >
(count(privilege[text()='OPEN_ACCT']) +
count(privilege[text()='CLOSE_ACCT']))"
diagnostics="Excess_Priv">The Privilege to Open and
Close Accounts should not be assigned to same role
     </sch:assert>
   <sch:diagnostics>
     <sch:diagnostic id="Excess_Priv">The errant role is:
<sch:value-of select="@role"/>
       </sch:diagnostic>
     </sch:diagnostics>
  </sch:rule>
  </sch:pattern>
```

The diagnostic error message generated by Schematron Validator is:

```
From pattern "Excess Privilege for a Role":
    Assertion fails: "The Privilege to Open and Close
Accounts should not be assigned to same role"  at
  /Bank_RBAC_Model[1]/RolePrivilegeAssignment[3]
  <RolePrivilegeAssignment role="CSR">...</> The errant
role is: CSR
```

Constraint 8: (Transaction Integrity Constraint): The right to perform certain operations should be assigned to more than one role in order to maintain the integrity of the transaction that is facilitated by this operation.

The constraint that the right to perform the operation of Loan Approval should be given to more than one role is specified as:

```
<sch:pattern name="Minimal Roles Roles required for an
operation">
  <sch:rule context="Bank_RBAC_Model">
    <sch:assert
test="count(RolePrivilegeAssignment[privilege[text()='APP
ROVE_LOAN']]) >=2"  diagnostics="Min_Role_Reqmt">A
Minimum of two roles is required for loan approval
</sch:assert>
   <sch:diagnostics>
     <sch:diagnostic id="Min_Role_Reqmt">The only role
now is: <sch:value-of
select="RolePrivilegeAssignment[privilege[text()='APPRO
VE_LOAN']]/@role"/>
       </sch:diagnostic>
     </sch:diagnostics>
  </sch:rule>
 </sch:pattern>
```

The diagnostic error message generated due to a violation of the policy constraint in our bank-enterprise authorization specification is:

```
From pattern "Minimal Roles Roles required for an
operation":
    Assertion fails: "A Minimum of two roles is required for
loan approval"  at
       /Bank_RBAC_Model[1]
      <Bank_RBAC_Model
xsi:noNamespaceSchemaLocation="A:\BankRBAC.xsd">..
.</> The only role now is: LNO
```

7. Related Work

A substantial work done in the area of policy specification and verification for authorization data [13,14] are based on the logic programming approach. A major disadvantage of the logic programming approach is that the authorization data embedded in the predicates and other logic programming artifacts cannot be easily extracted and mapped to the format that is amenable for instantiation within the access control frameworks of the various target platforms. XML-based frameworks for authorization specification certainly overcome this disadvantage. However the present XML-based approaches have limitations with respect to the type of policy constraints they can specify. The OASIS XACML [15] and IBM's XACL [16] are access control policy specification frameworks that are mainly geared towards securing XML documents and they do not provide support for representing traditional access controls such as DAC or MAC or enterprise access controls such as RBAC. Smith and Deng [17] have developed a DTD Schema of an RBAC Model and have left the verification of domain constraints to a separate administrative API. Further DTD Schemas in general have limitations with

respect to representation of even structural constraints (e.g. number of occurrences of an element) for an RBAC model and they cannot be used for representation of even rudimentary domain-specific policy constraints.

8. Scope for Further Work

We have provided a framework for validation of an enterprise authorization specification. The platform and language independent nature of the framework makes it an ideal candidate for incorporating this into the security data and policy specification module of enterprise security administration tools. With the availability of standardized API (e.g., DOM) based XML parsers, authorization specification data can be easily be mapped to the native formats required of various platform-specific access control mechanisms as is done in a class of tools called provisioning tools [18].

However, our constraint specification framework is not without its limitations. For example, it is not possible to formulate constraints that involve nested processing of data in two sets (e.g., verify consistency of privilege assignments for each pair of SSD roles). This limitation could be seen in the formulation of Constraint 4, where we had to identify the specific pair of conflicting users instead of formulating the constraint using a generalized list of all conflicted users. Further our constraint specification framework contains only static constraints but not dynamic constraints that make use of contextual information like time, location, process state etc. to enforce access restriction at run time. We are exploring the possibility of specifying such constraints through XSLT templates and then use XLST processors with various language bindings to automatically generate code in procedural languages like Java and C++ so as to incorporate them in the access enforcement module of the access control mechanism of target platforms.

9. References

[1] D.Ferraiolo, J.Cugini, and D.R.Kuhn. "Role Based Access Control (RBAC): Features and Motivations" Proc. 11th Annual Computer Security Applications Conference, December 1995.

[2] R.S. Sandhu, E.J.Coyne, H.L.Feinstein and C.E.Youman. "Role Based Access Control Models" IEEE Computer, vol 29, Num 2, February 1996, p38-47.

[3] XML 1.0, W3C Recommendation Feb '98, http://www.w3.org/XML/

[4] XML Schema Part 0: Primer W3C Recommendation, 2 May 2001 http://www.w3.org/TR/xmlschema-0/

[5] Document Object Model Technical Reports, http://www/w3.org/DOM/DOMTR

[6] R.Chandramouli, "Application of XML Tools for Enterprise-wide RBAC Implementation Tasks", Proc. Of 5th ACM workshop on Role-based Access Control, July 2000, Berlin, Germany.

[7] A.Schaad, "Role-based Access Control system of a European Bank: A Case Study and Discussion", Proc. Of 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001), Chantilly, VA, USA.

[8] Resource Description Framework (RDF), http://www.w3.org/RDF/

[9] Schematron - Pattern-based schema language, http://www.ascc.net/xml/resource/schematron/schematron.html

[10] D.Ferraiolo, R.Sandhu, S.Gavrila, D.R.Kuhn and R.Chandramouli, "Proposed NIST Standard for Role-based Access Control", ACM Trans. Inf.Syst.Security, Vol 4, Aug 2001, pp 224-274.

[11] http://www.xmlspy.com/download.html

[12] http://www.topologi.com/

[13] Jajodia, S., Samarati, P.,et al, "A Logical Language for Expressing Authorizations", IEEE Symposium on Security and Privacy, Oakland, CA, USA, 1997.

[14] Chomicki, J., Lobo, J., Naqvi, S. "A Logic Programming Approach to Conflict Resolution in Policy Management,", 7th Intl. Conf. On Principles of Knowledge Representation and Reasoning, Brackenridge, CO, USA, 2000.

[15] T. Moses, "The OASIS XACML Language Proposal", http://www/oasisopen.org/committees/xacml/docs

[16] M Kudo, S.Hada, "XML Access Control", "http://www.trl.ibm.com/projects/xml/xacl/xmlac-proposal.html, Oct 2000.

[17] Nathan N. Vuong , Geoffrey S. Smith , Yi Deng, Managing security policies in a distributed environment using extensible markup language (XML), Proceedings of the 16th ACM SAC2001 symposium on Applied computing March 2001, Las Vegas, Nevada, United States.

[18] Ferraiolo D.F, Kuhn D.R, Chandramouli R, "Role-Based Access Control", Artech House, April 2003.