published as: Wei Zhou, Christoph Meinel: A Policy Language for Integrating Heterogeneous Authorization Policies;
In Proceedings of the 4th International Conference on Grid Service Engineering and Management (GSEM 2007);
Leipzig, Germany, September 2007. pp. 9 - 23, ISBN: 978-3-88579-211-6

A Policy Language for Integrating Heterogeneous Authorization Policies

Wei Zhou, Christoph Meinel

Hasso-Plattner-Institute at University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
{wei.zhou, meinel}@hpi.uni-potsdam.de

Abstract. In order to manage and enforce multiple heterogeneous authorization policies in distributed authorization environment, we defined the root policy specification language and its corresponding enforcing mechanism. In a root policy, the involved users and resources can be defined in coarse or fine-grained. Each involved authorization policy's storage, trust management and enforcement can be defined independently. These authorization policies can be enforced in distributed way. Policy schemas, policy subschemas and policy hierarchies can deal with complex authorization scenarios. The context constraint component makes the root policy is a context-aware authorization system. On the other hand multiple root policies can cooperate together to complete more complicated authorization tasks.

Keywords: Access Control, Authorization Policy, XML, X.509, Attribute Certificate.

1 Introduction

Nowadays governments, businesses and organizations are finding that collaborations are increasingly critical to their success. The focus on sharing and protecting information is becoming increasingly acute. Authorization policy plays an important role in this era where computing resources of organizations with diverse privacy protection and information sharing requirements are increasingly connected together to carry out joint or common tasks. For information protection, different authorization systems and mechanisms are developed.

For instance, in response to the need to protect classified information, there are mechanisms to enforce Multi-Level Security (MLS) [1] policies, and in recognition of the needs of industry, Role-Based Access Control (RBAC) [2] mechanisms enforce commercial policies. For collaboration purposes, various new authorization systems and mechanisms have been proposed and developed recent years. Some of the well known systems are Akenti [3], Cardea [4], CAS [5], PRIMA [6], Permis [7] and VOMS [8]. So, access control mechanisms come in a wide variety of forms, each with their individual attributes, functions, and methods for configuring policy.

Unfortunately, in many application scenarios one size does not fit all, especially in the collaboration application environments such as a Grid system that is a virtual

In Proceedings of the 4th International Conference on Grid Service Engineering and Management (GSEM 2007), Leipzig, Germany, September 2007, pp 9-23.

organization composing several independent autonomous domains. In grid computing environment, each autonomous domain may have its own policies and may change its policies dynamically. Authorization in such a system needs to be flexible and scalable to support multiple authorization mechanisms and security policies, which suggest new challenges to the grid computing platforms. A further example is provided by laws concerning privacy issues. The security policy of an organization need combine internally specified constraints with externally imposed privacy regulations [9].

Even there uses only one authorization policy specification language, it is also possible to require an authorization system to combine multiple policies to complete some complex decision tasks. Consider a large organization composed of different departments and divisions, each of which can independently specify security policies; the global policy of the organization results from the combination of all these components. Finally, as security policies become more sophisticated, even within a single system it may be desirable to formulate the policy incrementally by assembling small, manageable, and independently conceived modules [17].

Some authorization policy languages, such as XACML [10], already provide comprehensive functionality to combine multiple policies if they are written in the same language. Some authorization policy languages, such as PERMIS X.500 PMI RBAC Policy [11], do not support multiple policies combination. So we need a policy language to specify multiple heterogeneous authorization policies combination and the corresponding enforcing mechanism. To our knowledge there is still no such kind of policy specification language. Motivated by this requirement, we developed a XML-based policy language for specifying heterogeneous policies combination and a mechanism to enforce these policies. This policy language is called root policy specification language, and a policy written in this language is called root policy. In this paper we will introduce the root policy structure and enforcing mechanism.

The remainder of this paper is organized as follows. Section 2 gives the major related work. Section 3 introduces the root policy authorization model. Section 4 describes the root policy language model. Section 5 provides a root policy enforcement mechanism. Section 6 investigates the root policy collaboration. Finally, Section 7 summarizes the results of this paper.

2 Related Work

Recent years there is considerable work on access control models and languages. Many approaches have been proposed to increase expressiveness and flexibility of authorization languages by supporting multiple policies within a single framework [12, 13, 14, 15, 16]. These proposals, while based on powerful languages able to express different policies, assume a single monolithic specification of the entire policy. Such an assumption does not fit many real-world situations, where access control might need to combine independently stated restrictions that should be enforced as one.

Since different organizations operate under different requirements for protecting their data, inevitably their security mechanisms do not share common principles, are not implemented in similar languages and do not run on compatible operating

platforms. This situation is recognized by both [17, 18]. They propose algebras for combining security policies with formal semantics. Complex policies are formulated as expressions of the algebras. These frameworks provide descriptions of policies that are language and implementation mechanism independent. Such descriptions can be examined for completeness, consistency, and unambiguity. Environment related policy composition also be considered by Siewe et al [19].

NIST initiated a project in pursuit of a standardized access control mechanism, referred to as Policy Machine (PM) that requires changes only in its configuration in the enforcement of arbitrary and organization specific attribute-based access control policies [20]. The PM's enforceable policies are combinations of policy instances. The core features of the PM are capable of configuring, combining and enforcing arbitrary attribute-based policies. PM categorizes users and objects and their attributes into policy classes, and transparently enforces these policies through a series of fixed PM functions. This may be a promise approach in future, but currently we need some more realistic solutions to resolve the already exists authorization problems.

XACML is a very rich and flexible language; users and security administrators can directly represent in XACML a large variety of authorization policies. XACML is becoming popular these years. More and more systems adopt XACML as their authorization language. However, XACML has not been built to manage security in large distributed systems in which virtual organizations are dynamically built with the collaboration of multiple independent subjects sharing their resources [21].

Multipolicy Authorization Framework for Grid Security [22] is the most relevant work to our work. Basing on the special security needs of the Grid computing, they constructed an authorization framework in the Globus Toolkit 4 [23] that can support multiple authorization policies. For each existing authorization policy, the framework constructs a Policy Decision Point (PDP) for evaluating that kind of policy. There is a Master PDP that is responsible for coordinating other PDPs, combining the decisions returned by each PDP and renders a final decision. The PDPs managed by a Master PDP are specified in a security configuration file. However their approach does not touch such as policy storage, interaction among PDPs and multiple Master PDPs collaboration. There is not policy language for specifying and enforcing these PDPs, and simply adds policy evaluator class names into a configure file. So, all PDPs are applicable to all authorization requests.

3 Root Policy Authorization Model

Root policy is used to manage and enforce multiple heterogeneous authorization policies. A root policy authorization system consists of a root policy and a root policy evaluator. The root policy evaluator acts the role of a PDP. It reads a root policy in, evaluates input authorization requests, and then renders authorization decisions.

3.1 Root Policy Data Flow Model

The architecture of root policy framework adopts the XACML authorization model. This model uses an entity named Context Handler to separate Policy Enforcement

Point (PDP) and Policy Decision Point (PEP). So an application system can communicate with a XACML evaluator or other policy evaluator such as our root policy evaluator through the context handler. This is the major reason that we adopt this authorization model. The root policy authorization model is shown in Fig. 1.

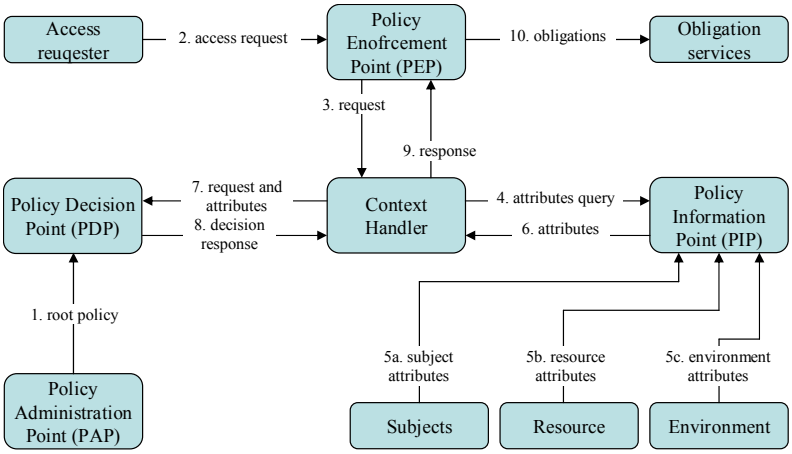

Fig. 1. Root policy authorization model.

The root policy authorization model mainly contains PEP, PDP, Context Handler, Policy Information Point (PIP) and Policy Administration Point (PAP). PEP performs access control by making decision requests and enforcing authorization decisions. PAP creates root policy that is used by a root policy PDP.

PIP collects information about the request subject, related resource and the environment. It's useful to separate this collection process into its own module so that different authorization algorithms and policies can be configured with the same collection process. Examples of PIPs are the VOMS [8] PIP, which parses a caller's VOMS credential for attributes; and the Shibboleth [24] PIP, a remote callout that retrieves Security Assertions Markup Language (SAML) [25] attributes; and the Permis [7] PIP, which parses a caller's X.509 attribute certificates [26] for user roles.

Context handler constructs root policy canonical requests context based on the request sent by the PEP and the additional attributes obtained from the PIP, and then presents them to the PDP. PDP evaluates the root policy and renders an authorization decision. Context handler converts the authorization decisions from PDP to the native format supported by the PEP. The PEP fulfills the obligations and either permits or denies the access request according to the decision of PDP.

3.2 Root Policy Context

Root policy is intended to be suitable for a variety of application environments. The core language is insulated from the application environment by the root policy

context, as shown in Fig. 2. The root policy context is defined with a *root policy*, *root policy evaluator*, *root policy request context* and *root policy response context*.

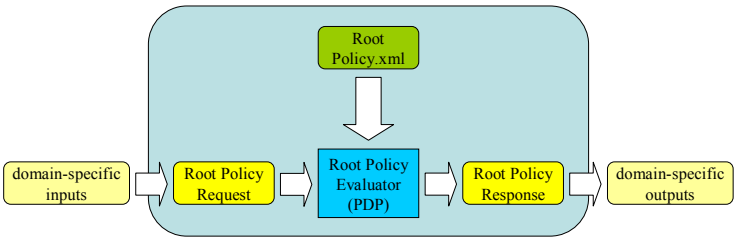

Fig. 2. Root policy context.

The root policy is written in XML. It describes which authorization policies are used, where to get them, and how to combine them to reach a final authorization decision. Root policy evaluator is used to evaluate a root policy. It is also called root policy PDP. A root policy PDP makes authorization decisions through combining other concrete authorization policy evaluators' authorization results.

In one root policy there may involve several different types of authorization policies and their corresponding policy evaluators. Each kind of policy evaluators may need different input data format. The root policy request context is a data structure that can provide different request information used to making particular decision requests for different policy evaluators, and new request information can be easily added into it. It is the responsibility of an authorization policy evaluator to create a concrete decision request according to the root policy request context.

The evaluation result of a root policy can be "Permit", "Deny", "Indeterminate" or "NotApplicable" that is carried by the root policy response context. It is the responsibilities of the context handler to convert the domain-specific inputs to root policy requests and convert root policy responses to the domain-specific outputs.

3.3 Root Policy Trust Management

Privilege management infrastructure (PMI) was specified by the ITU-T and ISO/IEC [27]. The main function of PMI is in providing a strong authorization after the authentication has taken place. It has a number of similarities with PKI [28]. The basic data structure in a PMI is a X.509 attribute certificate (AC) [26]. Like public key certificate (PKC) strongly binds a public key to its subject, AC strongly binds a set of attributes to its holder. Attribute certificates have been designed to be used in conjunction with identity certificates, i.e. PMI and PKI infrastructures are linked by information contained in the ACs and PKCs. For example the holder field in an AC contains the serial number and issuer of a PKC.

In a PMI, the entity that digitally signs an AC is called Attribute Authority (AA). The trusted root of a PMI is called Source of Authority (SOA). A SOA may delegate its powers of authorization to subordinate AAs. Subordinate AAs may also delegate their powers of authorization to further subordinate AAs. Then an AA hierarchy can

be established. When a user's authorization permissions need to be revoked, an AA will issue an Attribute Certificate Revocation List (ACRL). There are two primary models for distribution of ACs: the "push" and "pull" models. ACs may be used with various security services, including access control, data origin authentication, and non-repudiation. In our work ACs are used to store authorization policies.

4 Root Policy Language Model

The root policy language model is shown in Fig. 3. The main components of the model are *subject domain*, *resource domain*, *context constraint*, *policy*, *policy hierarchy*, *policy schema* and *policy subschema*. They are described as follows.

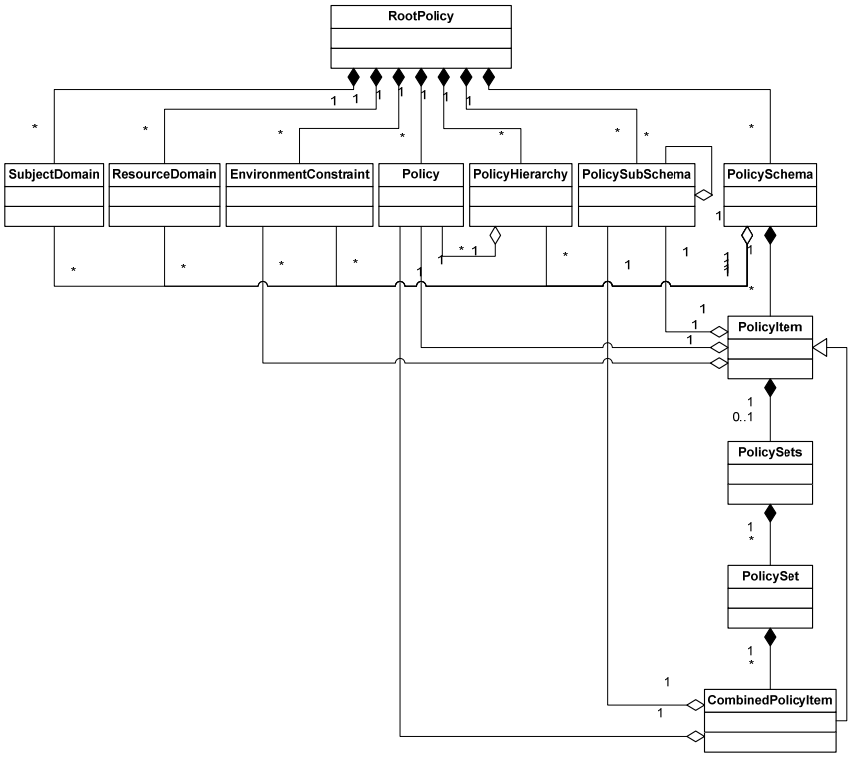

Fig. 3. Root policy language model.

4.1 Subject Domain

SubjectDomain specifies the domain of users who may be granted within the overall root policy. Each domain is specified as a collection of LDAP sub-trees, using Include

and Exclude statements. The Include statement specifies the LDAP DN of the root node of a subject domain, and the Exclude statement specifies which subordinate sub-trees to be excluded from the domain. Using a null LDAP DN in an Include statement specifies the domain of all users in the world.

An example of directory information tree is shown in Fig. 4. The example of subject domains specification below specifies the employees of company ABC, excluding the Germany branch.

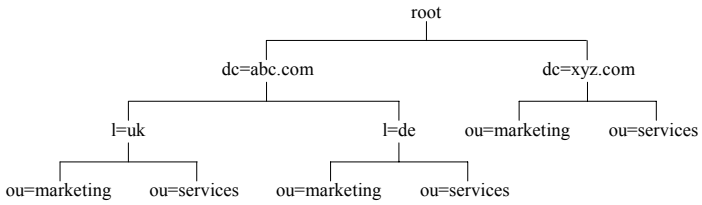

Fig. 4. Example of subject domain directory information tree.

```
<SubjectDomains>
  <SubjectDomain SubjectDomainID=" CompanyABCEmployees ">
    <Include>
      <LDAPDN>dc=abc.com</LDAPDN>
    </Include>
    <Exclude>
      <LDAPDN>l=de, dc=abc.com</LDAPDN>
    </Exclude>
  </SubjectDomain>
</SubjectDomains>
```

4.2 Resource Domain

ResourceDomain specifies the resource domain covered by this root policy. Resource domains are specified as LDAP DN sub-trees, using Include, Exclude and ObjectClasses statements. The Include statement specifies the LDAP DN of the root node of a domain, and the Exclude statement specifies subordinate sub-trees to be excluded from the domain. Using a null LDAP DN in an Include statement specifies the domain of all resources in the world. A domain may optionally be refined by specifying a set of object classes. An object class is a general description of an object as opposed to the description of a particular object. For instance, the object class CustomerInformation represents the customer information of a company. Only resources with the full set of object classes are included in the domain. A null set of object classes implies all resources in the domain are included.

The following example comprises a resource domain as shown in Fig. 4, which specifies all the customer information in the company ABC.

```
<ResourceDomains>
  <ResourceDomain ResourceDomainID="CompanyABCCustomers">
    <Include>
      <LDAPDN>ou=services, l=uk, dc=abc.com</LDAPDN>
```

```
    </Include>
    <ObjectClasses>
      <ObjectClass>CustomerInformation</ObjectClass>
    </ObjectClasses>
  </ResourceDomain>
</ResourceDomains>
```

4.3 Context Constraint

ContextConstraint specifies that certain context attributes must meet certain conditions to permit an authorization policy or an authorization policy schema to be executed. A context constraint is defined through the terms *context attribute*, *context function* and *context condition*:

- *Context attribute* represents a certain property of the context whose actual value might change dynamically, e.g. time, date, location and etc.
- *Context function* is a mechanism to obtain the current value of specific context attribute. For example, the function *getDate* returns the current date. One or more context functions are encapsulated into a context observer. For example, functions *getDate*, *getTime* and *getIP* can be organized into the observer *LocalHostObserver*.
- *Context condition* is a predicate that consists of an operator and two or more operands. The first operand represents a certain context attribute, while the other operands may be either context attributes or constant values. Each context attribute is replaced with a constant value by using the corresponding context function prior to the evaluation of the respective condition.
- *Context constraint* is a class clause containing one or more context conditions. It is satisfied if and only if all its context conditions are satisfied.

A conditional policy or policy schema is associated with one or more context constraints and grants to be used for making access control decisions if and only if each corresponding context constraint evaluates to "true". The following example specifies a context constraint of WorkingTime that specifies the working days are from Monday to Friday in a week.

```
<ContextConstraints>
  <ContextConstraint ContextConstraintID="WorkingTime">
    <ContextCondition ContextConditionID="DaysOfWeek">
      <Operator DataType="WeekDay">in</Operator>
      <LeftOperand>
        <Observer>LocalHostObserver</Observer>
        <Function>GetWeekDay</Function>
      </LeftOperand>
      <RightOperand>
        <Parameter>Monday;Tuesday;Wednesday;Thursday;Friday</Parameter>
      </RightOperand>
    </ContextCondition>
  </ContextConstraint>
</ContextConstraints>
```

4.4 Policy

Policy specifies the authorization policy used in a root policy. These polices' storage can be distributed. In order to ensure the policies gotten are the real ones at runtime, X.509 attribute certificates are used to hold authorization policies. The items defined to specify an authorization policy are described as follows.

- *OID* specifies the unique object identifier of an authorization policy. This value is also used to extract the corresponding policy from a policy attribute certificate.
- *ValidityPeriod* specifies the valid time of a policy. The actual validity time takes the intersection of the policy validity time the AC validity time.
- *Critical* specifies how to deal with this policy after it is expired. If its value is "TRUE", then all the authorization requests related to this policy will be denied.
- *Evaluator* specifies the class that is used to evaluate this policy.
- *ACURI* is the unique name used to retrieve the policy AC.
- *ACRLURI* specifies the attribute certificate revocation list (ACRL).
- *PKCURI* is the unique name used to retrieve the PKC used to verify the policy AC.
- *CRLURI* specifies the PKC revocation list (CRL).
- *RealizedBy* specifies where an authorization request should be further sent when it needs to be evaluated at a remote site.

The following example specifies two policies. The policy "Permis_Policy" is evaluated at the local site. The policy "XACML_Policy" is evaluated in a remote site, and the requests are sent to the remote site through SOAP messages.

```
<Policies>
  <Policy PolicyID="Permis_Policy">
    <OID>1.2.826.0.1.3344810.1.1.13.1</OID>
    <ValidityPeriod Start="2006-06-01T00:00:00" End="2007-08-31T23:59:59" />
    <Critical>TRUE</Critical>
    <Evaluator>PDP_PERMIS</Evaluator>
    <ACURI>ldap://localhost:389/cn=PermisPolicy1,ou=Security,dc=uni-
trier.de</ACURI>
    <ACRLURI>http://localhost:8080/Trier/ACs/ACRL/</ACRLURI>
    <PKCURI>ldap://localhost:389/cn=aaRSA1024Trier,ou=Security,dc=uni-
trier.de</PKCURI>
    <CRLURI>http://localhost:8080/Trier/PKCs/CRL/</CRLURI>
  </Policy>
  <Policy PolicyID="XACML_Policy">
    <OID>1.2.826.0.1.3344810.1.1.15.1</OID>
    <Critical>TRUE</Critical>
    <RealizedBy>http://remotehost:8080/root-policy/coordinator</RealizedBy>
  </Policy>
</Policies>
```

4.5 Policy Hierarchy

PolicyHierarchy specifies the inheritance relation among policies. At the top of the hierarchy, the policy is the most general to all other policies. Policies near the bottom of the hierarchy provide more specialized specification. Except the policy at the root node, any other policy has one and only one direct superior policy. So the policy hierarchy is a simple tree. An example of policy hierarchy is shown in Fig. 5.

Policy hierarchy caters the requirement of large organization composed of different divisions, each of which can independently specify security policies; the global policy of the organization results from the combination of all these policies. Similar requirement also exists in virtual organization that needs to combine the virtual organization scope policies and the policies from the participant organizations.

When an authorization request is checked by a policy that is in a policy hierarchy, all the policies from the given policy to the root node policy may be checked. Each policy hierarchy is associated a policy combining algorithm used to combine these policies. They are "deny-overrides" and "permit-overrides". The following example specifies a policy hierarchy shown in Fig. 5, and the associated policy hierarchy combining algorithm is "permit-overrides".

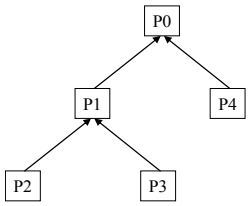

Fig. 5. Example of policy hierarchy.

```
<PolicyHierarchies>
  <PolicyHierarchy PolicyHierarchyID="Policy_Hierarchy"
PolicyCombiningAlgID="policy-combining-algorithm:permit-overrides">
    <PolicyReference PolicyID="P0">
      <SubPolicyReference PolicyID="P1" />
      <SubPolicyReference PolicyID="P4" />
    </PolicyReference>
    <PolicyReference PolicyID="P1">
      <SubPolicyReference PolicyID="P2" />
      <SubPolicyReference PolicyID="P3" />
    </PolicyReference>
    <PolicyReference PolicyID="P2" />
    <PolicyReference PolicyID="P3" />
    <PolicyReference PolicyID="P4" />
  </PolicyHierarchy>
</PolicyHierarchies>
```

4.6 Policy Schema

PolicySchema specifies what kinds of authorization requests should be checked by which policies, and how these policies are combined together to make access control decisions. As shown in Fig. 3, the policy schema contains five data elements, they are described as follows.

SchemaSubjectDomains specifies the domains from which the subjects of authorization requests are valid. *SchemaResourceDomains* specifies the domains in which the objects of authorization requests are valid. A policy schema is applicable only if the authorization request satisfied the schema subject domains, schema

resource domains and its associated context constraints are evaluated to “true”. *SchemaPolicyHierarchies* specifies which policy hierarchies should be considered.

PolicyItem is the basic data element used to describe policy relations. Each PolicyItem holds a policy reference or a policy subschema reference. Its related policy or policy subschema is organized in *CombinedPolicyItem*. The combined policy items are organized into *PolicySet* elements. One PolicyItem can have multiple PolicySet elements that are organized into *PolicySets* element. The algorithms used to combine multiple CombinedPolicyItems or PolicySets are “deny-overrides” and “permit-overrides”. The relation between PolicyItem and PolicySets is conjunction. A policy reference held by a CombinedPolicyItem can also be held by a policy item, through this way, any relation among these policies can be specified. A policy item can be associated with context constraints that are used to limit the usage of the policy items based on the context information.

In each policy schema there is a policy item defined as the start point from which authorization requests are checked. The evaluation result of policy schema can be “Permit”, “Deny”, “NotApplicable” or “Indeterminate”. It is also possible that multiple policy schemas satisfy one authorization request, then the policy schema combining algorithm will be used to combine the multiple policy schemas. The policy schema combining algorithm can either be “deny-overrides” or “permit-overrides”.

The following example shows an authorization policy schema definition. In this example, the policy schema defines two schema subject domains and one schema resource domain. The schema also defines two policy hierarchies. A context constraint is also associated to this policy schema. The start point of this policy schema is the policy item “Permis_Policy_1”. There two policies are related to this policy item and organized into one policy set.

```
<PolicySchemas SchemaCombiningAlgID="schema-combining-algorithm:permit-
overrides">
  <PolicySchema PolicySchemaID="B2BMCPolicySchema1">
    <StartPolicyItem PolicyItemID="Permis_Policy_1" />
    <SchemaSubjectDomains>
      <SubjectDomainReference SubjectDomainID="TrierEmployees" />
      <SubjectDomainReference SubjectDomainID="PotsdamEmployees" />
    </SchemaSubjectDomains>
    <SchemaResourceDomains>
      <ResourceDomainReference ResourceDomainID="B2BMCFiles" />
    </SchemaResourceDomains>
    <SchemaPolicyHierarchies HierarchyCombiningAlgID="hierarchy-combining-
algorithm:deny-overrides">
      <PolicyHierarchyReference PolicyHierarchyID="Policy_Hierarchy_1" />
      <PolicyHierarchyReference PolicyHierarchyID="Policy_Hierarchy_2" />
    </SchemaPolicyHierarchies>
    <SchemaPolicyRelations>
      <PolicyItem PolicyItemType="Policy" PolicyItemID="Permis_Policy_1">
        <PolicySets PolicySetCombiningAlgID="policy-set-combining-
algorithm:permit-overrides">
          <PolicySet PolicyItemCombiningAlgID="policy-item-combining-
algorithm:permit-overrides">
            <CombinedPolicyItem PolicyItemType="Policy"
PolicyItemID="XACML_Policy_1" />
            <CombinedPolicyItem PolicyItemType="Policy"
PolicyItemID="XACML_Policy_2" />
          </PolicySet>
```

```
        </PolicySets>
      </PolicyItem>
    </SchemaPolicyRelations>
    <Context>
      <ContextConstraintReference ContextConstraintID="WorkingTime" />
    </Context>
  </PolicySchema>
</PolicySchemas>
```

4.7 Policy Subschema

PolicySubSchema is a simplified policy schema. It can be invoked by policy schemas or other policy subschemas. The major difference between policy schema and policy subschema is that the subschema does not specify the schema subject domains and schema resource domains. This characteristic makes policy subschemas can be invoked by different policy schemas that have different applicable scopes.

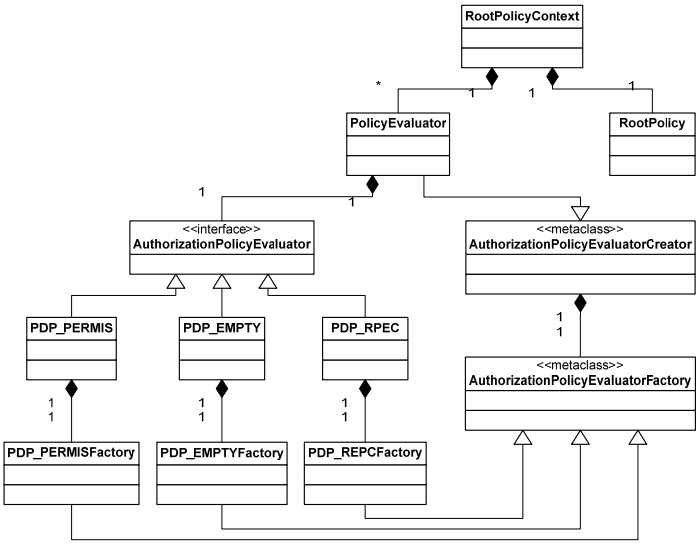

Fig. 6. Root policy evaluator.

5 Root Policy Enforcement

The components used to enforce a root policy are shown in Fig. 6. The class RootPolicyContext is the container that holds a RootPolicy object and the authorization policy evaluators used to evaluate corresponding authorization policies specified in the root policy. Each policy evaluator is an instance of the class PolicyEvaluator, and is used to evaluate a concrete authorization policy. In order to

initialize a policy evaluator, the policy evaluator class name and policy filename are passed into the PolicyEvaluator class constructor. Each policy evaluator can be seen as an independent PDP that is managed by the root policy PDP.

A complicated access control system may involve multiple types of policies, and new type of policies may also arise over a period of time. So the root policy evaluator is designed for policy agnostic. Each kind of policies has its own evaluator, and different kinds of policy evaluators have the same interface so that the root policy evaluator can treat them in a unified way. New policy evaluator can be dynamically added or removed from the system at runtime. In Fig. 6 the PDP_PERMIS, PDP_EMPTY and PDP_RPEC are three kinds of policy evaluators. The PDP_PERMIS is the evaluator for evaluating X.509_PMI_RBAC_Policy policies [11]. In a root policy system, we can define any number of such kinds of policy evaluators. The PDP_EMPTY and PDP_RPEC are used to deal with empty policy items and remote root policy callings, respectively. The PolicyEvaluator manages them through the interface AuthorizationPolicyEvaluator. Each policy evaluator is implemented separately and independently. The only restriction is all of them must share a similar interface. The corresponding Java interface is defined as:

```
public interface AuthorizationPolicyEvaluator {
  public int evaluate(RequestContext requestContext);
}
```

This method must be implemented in a concrete authorization policy evaluator, through which decision request contexts are passed into the policy evaluator and then the evaluation results are returned. Each policy evaluator is an instance of the class PolicyEvaluator. The PolicyEvaluator inherits from the abstract class AuthorizationPolicyEvaluatorCreator that is responsible for creating an AuthorizationPolicyEvaluator object.

6 Root Policy Collaboration

Multiple root policies can collaborate to finish some common tasks. These root policies can coexist in one domain or different domains. The implementation of root policy collaboration is through the way that the authorization policy specified in one root policy is realized by another root policy. In this case the policy specification field RealizedBy provides a URL pointing to an endpoint where the policy will be enforced. All the decision requests related to this policy will be sent to this endpoint. Every root policies and the authorization policies specified in these root policies have a unique identification (OID) in the whole scope that these policies are applied. In the case that one policy is realized by another root policy, its OID is another root policy OID. One example of these relationships is shown in the Fig.7.

In this example there are four root policies. In the Root Policy1 there define three authorization policies. Among of them the Policy2 is realized by the Root Policy2. In the Root Policy2 there define two authorization policies, both of them are realized by some other root policies, i.e. the Policy4 is realized by the Root Policy 3 and Policy5

is realized by the Root Policy4. Through this way, one decision request check can be propagated among many root policies.

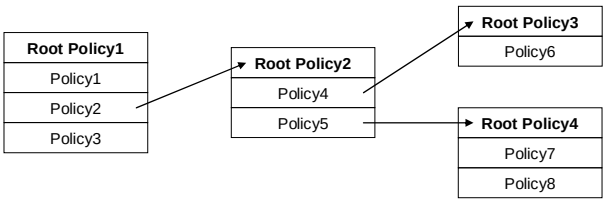

Fig. 7. Example of root policy collaboration.

7 Conclusion

Root policy specification language is a XML-based language for specifying heterogeneous authorization policy management and enforcement. It allows security administrators to freely define the involved authorization policy's storage, trust management and enforcement independently. New kinds of policy evaluators and policies can be dynamically added into the system. With the help of policy schemas, policy subschemas and policy hierarchies, complex authorization policy relations can be easily defined. The context constraint component makes the root policy is a context-aware specification language that is important for specifying virtual organization management, in which the involved users and resources are dynamically changed. On the other hand multiple root policies can cooperate together to complete more complicated authorization tasks.

References

1. D. Bell and La Padula, "Secure computer systems: unified exposition and MULTICS". Report ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, March 1976.
2. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control", ACM Transactions on Information and System Security, vol. 4, pp. 224-274, Aug. 2001.
3. M. Thompson, A. Essiari, S. Mudumbai, "Certificate-based Authorization Policy in a PKI Environment", ACM Transactions on Information and System Security (TISSEC), Volume 6, Issue 4 (Nov. 2003) pp 566-588
4. R. Lepro, "Cardea: Dynamic Access Control in Distributed Systems", NASA Technical Report NAS-03-020, November 2003.
5. L. Pearlman, C. Kesselman, V. Welch, I. Foster and S. Tuecke, "The Community Authorization Service: Status and Futures". Computing in High Energy Physics (CHEP03), 2003.
6. M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi and S. Shah, "The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments", 4th Int. Workshop on Grid Computing - Grid 2003, 17 November 2003, Phoenix, AR, USA.

7. D.W. Chadwick and A. Otenko, “The PERMIS X.509 role based privilege management infrastructure”, Future Generation Computer Systems, Volume 19, Issue 2, February 2003, Pages 277-289.
8. R. Alfieri, R. Cecchini, V. Ciaschini, L. dell’Agnello, Á. Frohner, K. Lőrentey and F. Spataro, “From gridmap-file to VOMS: managing authorization in a Grid environment”, Future Generation Computer Systems, 21(4), pp. 549-558, 2005.
9. D. Banisar and S. Davies, “Privacy & Human Rights—An International Survey of Privacy Laws and Developments”, EPIC, 1999.
10. XACML and OASIS Security Services Technical Committee, “eXtendible Access Control Markup Language (xacml) committee specification 2.0”, Feb 2005.
11. D.W. Chadwick, A. Otenko, “RBAC Policies in XML for X.509 Based Privilege Management”, SEC 2002, Egypt, May 2002.
12. H. Hosmer, “The multipolicy paradigm”, in Proceedings of the Fifteenth National Computer Security Conference (Baltimore, Oct.), 409–422, 1992.
13. T. Woo and S. Lam, Authorizations in distributed systems: A new approach. J. Comput. Sec. 2, 2,3, 107–136, 1993.
14. E. Bertino, S. Jajodia and P. Samarati, “A flexible authorization mechanism for relational data management systems”, ACM Trans. Inf. Syst. 17, 2 (April), 101–140, 1999.
15. N. Li, J. Feigenbaum and B. Grosof, “A logic-based knowledge representation for authorization with delegation”, in Proceedings of the Twelfth IEEE Computer Security Foundations Workshop (Mordano, Italy, June), 162–174, 1999.
16. S. Jajodia, P. Samarati, M. Sapino and V. Subrahmanian, “A unified framework for supporting multiple access control policies”, ACM Trans. Database Syst. 26, 2 (June), 214–260, 2001.
17. P. Bonatti, S. Vimercati, and P. Samarati, “An Algebra for Composing Access Control Policies”, ACM Transaction on Information and System security, 5(1):1-35, February 2002.
18. D. Wijesekera and S. Jajodia, “A Propositional Policy Algebra for Access Control”, ACM Transactions on Information and System Security, 6(2):286–325, May 2003.
19. F. Siewe, A. Cau and H. Zedan, “A Compositional Framework for Access Control Policies Enforcement”, in proceedings of the ACM FMSE’03, Washington, DC, USA, pp. 32-42, October 2003.
20. D. F. Ferraiolo, S. Gavrila, V. Hu, D. R. Kuhn, “Access control policy management: Composing and combining policies under the policy machine”, in Proceedings of the SACMAT’05, Stockholm, Sweden, pp. 11-20, 2005.
21. P. Mazzoleni, E. Bertino, B. Crispo, S. Sivasubramanian, “XACML policy integration algorithms: not to be confused with XACML policy combination algorithms!”, in Proceedings of the SACMAT’06, Lake Tahoe, California, USA, pp. 219-227, June 2006.
22. B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan and T. Freeman. “A Multipolicy Authorization Framework for Grid Security”, in Proceedings of the Fifth IEEE Symposium on Network Computing and Application, Cambridge, USA, pp. 269-272, July 2006.
23. The Globus Project: http://www.globus.org/.
24. The Shibboleth project, http://shibboleth.internet2.edu/.
25. Security Assertion Markup Language (SAML) v1.0 - OASIS Standard, 5 November 2002 - http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf.
26. S. Farrell, R. Housley, An Internet Attribute Certificate Profile for Authorization, Internet-draft April 2002, http://www.ietf.org/rfc/rfc3281.txt.
27. ITU-T Rec. X.509 ISO/IEC 9594-8, The Directory: Public-key and Attribute Certificate Frameworks, May, 2001.
28. R. Housley, W. Ford, W. Polk, D. Solo, Internet X.509 Public Key Infrastructure Certificate and CRL Profile, January 1999, http://www.ietf.org/rfc/rfc2459.txt.