RC24875 (W0910-079) October 15, 2009
Computer Science

IBM Research Report

Extending the CIM-SPL Policy Language with RBAC for Distributed Management Systems in the WBEM Infrastructure

Li Pan
Department of Electronic Engineering
Shanghai Jiao Tong University
China

Jorge Lobo, Seraphin Calo
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Extending the CIM-SPL Policy Language with RBAC for Distributed Management Systems in the WBEM Infrastructure

Li Pan
Department of Electronic Engineering
Shanghai Jiao Tong University, China
panli@sjtu.edu.cn

Jorge Lobo
IBM T. J. Watson Research Center
USA
jlobo@us.ibm.com

Seraphin Calo
IBM T. J. Watson Research Center
USA
scalo@us.ibm.com

Abstract— *In spite of the large effort behind the development of the WBEM and CIM standards for the management of distributed systems, there has been very little work addressing security in those standards. In this paper we present a Role-based Access Control (RBAC) policy language to render fine-grained access control policies for WBEM and CIM. The language is an extension of CIM-SPL, a preliminary DMTF policy language standard. The CIM-SPL RBAC extension fully complies with the WBEM standards. Access control policies can be specified for CIM object constructs according to the standard NIST RBAC model as well as with an extended model adapted for CIM. An implementation framework for the CIM-SPL RBAC in the OpenPegasus WBEM infrastructure is also presented to demonstrate its usability. Some design choices and implementation issues are discussed in detail. This framework provides an end to end solution to deploy a policy-based RBAC mechanism in the WBEM infrastructure.*

1. Introduction

With the increasing complexity of IT infrastructures, realizing a distributed management system across disparate technologies and platforms becomes a costly and laborious task. To solve this problem, the Distributed Management Task Force (DMTF) proposed Web-Based Enterprise Management (WBEM) standards. The goal of these standards is to develop a unified way to describe, store and manipulate the resources to be managed and to provide a standard communication protocol for distributed management systems. Currently, more and more IT venders are tending to support WBEM standards in their products. There are also several open source projects dedicated to providing an implementation infrastructure for WBEM standards (e.g., OpenPegasus [1]). Based on the WBEM infrastructure, domain-specific management systems in various areas, including networks, storage systems and severs, can be conveniently established. However, although access control is a crucial security issue for such systems, few WBEM standards refer to this issue, and only a preliminary mechanism is provided in the OpenPegasus implementation. There is no satisfactory way to render a fine-grained access control solution in the WBEM infrastructure.

Another important research effort led by both academia and industry to facilitate the system management task by is the development of policy-based management technology [2]. There has been also interest in the integration of policy-based management technology into the WBEM infrastructure. To this aim, a novel policy language, named CIM-SPL, is proposed to simplify the definition and application of policy rules governing the resources stored in Common Information Model (CIM) constructs. CIM-SPL has been approved by the DMTF as a suggested policy language in the WBEM infrastructure [3]. However, the current version of CIM-SPL has been designed to specify general purpose management policy rules. It does not directly define access control policies.

Motivated by the above observations, the authors extended the CIM-SPL language to render a policy-based access control solution for the WBEM infrastructure. The design goal is to provide an easy way to specify authorization policy rules following the standard Role-based Access Control (RBAC) model for managed resources depicted by CIM objects. Furthermore, additional features, such as conditional role-to-permission assignments and attribute-based user-to-role assignments are added in the policy language to provide maximum flexibility for system administration. The CIM-SPL RBAC extension proposed in this paper inherited all the merits of the original language, i.e., it is compliant with CIM constructs and consistent with corresponding WBEM standards. An implementation framework for the CIM-SPL RBAC extension in OpenPegasus is also presented to demonstrate its usability, and then some design choices and implementation issues are discussed in detail.

The remainder of this paper is structured as follows: Section 2 presents background on the WBEM infrastructure and the CIM-SPL policy language; Section 3 provides details on the model, syntax and semantics of our CIM-SPL RBAC extension; Section 4 provides examples of role-based access control policy rules defined in the CIM-SPL RBAC extension to illustrate its features; Section 5 describes the proposed OpenPegasus implementation framework; and, finally, Section 6 concludes the paper and points to future work.

2. Background

2.1. WBEM Infrastructure

Although the WBEM standards are defined at an architectural level to provide greater flexibility, when they are used the general WBEM Infrastructure can be illustrated as in Figure 1.

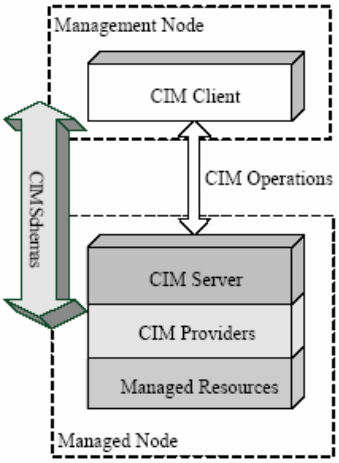

Figure 1. WBEM Infrastructure

In the above WBEM infrastructure, the communication information between the Management Node and the Managed Node is standardized and can be transported over HTTP. The managed resources are defined according to the CIM Specification. A management application, which acts as a CIM Client, issues CIM Operation requests and processes received CIM Operation responses. The CIM Server receives and processes these CIM Operation requests. Most of them will be forwarded to a specific CIM Provider. The CIM Provider translates the CIM formatted requests into resource specific operations and translates resource-specific responses into CIM formatted responses which will eventually be returned to the CIM Client via the CIM Server. All the management activities are enforced through CIM Operation requests which invoke one or more methods. These methods are either intrinsic operations defined by the specification to model general CIM operations or extrinsic operations defined as methods of specific CIM classes [4]. The intrinsic operations can be grouped by their functionalities as showed in Table 1.

Table 1. CIM Intrinsic Operations

Functional Group	CIM Intrinsic Operations
Basic read	GetClass, EnumerateClasses, EnumerateClassNames, GetInstance, EnumerateInstances, GetProperty EnumerateInstanceNames
Basic Write	SetProperty
Schema Manipulation	CreateClass, ModifyClass, DeleteClass
Instance Manipulation	CreateInstance, ModifyInstance, DeleteInstance
Association Traversal	Associators, AssociatorNames, References, ReferenceNames
Query	ExecQuery
Qualifier Declaration	GetQualifier, SetQualifier, DeleteQualifier, EnumerateQualifier

2.2. CIM-SPL

CIM-SPL is a declarative policy language defined to specify condition-action structured policies governing CIM objects. Each CIM-SPL policy is written under the scope of a single CIM Object referred to as the anchor object of the policy. All other CIM Objects referenced by a CIM-SPL policy must be accessible by traversing CIM associations starting from the anchor object of the policy. A general CIM-SPL policy follows the syntax structure shown in Figure 2.

```
import <MOF Name>::<CIM Class Name>:<Condition>
policy{
   declaration { <constant and macros> }
   condition { <Java-like boolean expression> }
   decision { <action workflow> }
}
```

Figure 2. CIM-SPL Policy Rule Syntax

The boldface words are resrved key words of the language (we follow this method of representation in the remainder of the paper). The import statement defines the class of the anchor object for the policy. The declaration section is optional and defines macros and constants to simplify notation in the condition. The condition and decision sections correspond respectively to the "if and then" clauses of a policy rule. A unique feature of CIM-SPL is that it has a set of built-in operators that allow traversal of CIM associations collecting required system components during a policy evaluation. Besides these built-in operators, CIM-SPL has all conventional arithmetic, boolean, and casting operators along with operators for

string and datetime manipulation. It also supports all intrinsic data types defined by the CIM Meta Schema and one-dimensional arrays of an intrinsic type. These characteristics make CIM-SPL not only compatible with CIM but also powerful for policy representation.

Consistent with the CIM Policy Model, policies can be organized into groups in CIM-SPL. The definition of a policy group follows the syntax shown in Figure 3. A policy group has an optional association. The traversal of this association is used for gathering required data during the evaluation of the policy group. This simple but powerful mechanism significantly simplifies the management of policies for hierarchical CIM objects.

```
import<MOF Name>::<CIM Class Name>:<Condition>
strategy <Exiecution Strategy>
declaration{ . . . }
policy{ . . . }:<Priority>
. . .
policy{ . . . }:<Priority>
policyGroup:<Association Class>
            (<Property One>,<Property Two>)
             { . . . }:<Priority>
```

Figure 3. CIM-SPL Policy Group Syntax

3. Extending CIM-SPL with RBAC

Generally, an access control policy specifies on what conditions to grant permission to a subject to perform actions on a protected object. In the WBEM context discussed in section 2.1, the protected objects are managed resources contained in CIM data structures. The controlled actions are management activities requested by the CIM Client to manipulate CIM Objects. They are either intrinsic or extrinsic CIM Operations. Since the Role-based Access Control (RBAC) model has great power for representing various authorization policies, our CIM-SPL extension is defined so as to provide a complete approach for using RBAC policies in WBEM-based management applications. Our design objectives are that: the RBAC policies should be easy for a policy author to write;, the extension is CIM compliant; and, the language style is consistent with the original CIM-SPL.

To achieve the first design objective, a close correspondence between access control policy rules and the standard NIST RBAC model [5] is established, which is illustrated in Figure 4.

The bottom part of the figure is a simplified RBAC model including the main elements such as users, roles, permissions, sessions and constraints. As shown in the gray blocks at the top of the figure, three new types of Policy Rules are proposed in our CIM-SPL RBAC extension; these are Permission Rules, PA Rules and UA Rules. Each of them represents one or more important components in the RBAC model. The condition-action style Policy Rule in the original CIM-SPL is then called the Obligation Policy Rule, since it expresses the semantics that when the defined conditions are satisfied specific management activities should be enforced.

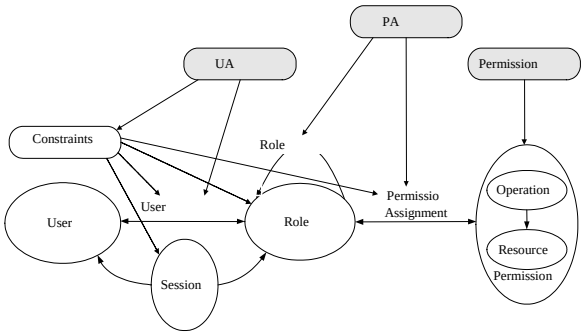

Figure 4. Mapping CIM-SPL Access Control Policy Rules to the NIST RBAC Model

A Permission Policy Rule is used to define the approval of CIM operations on managed resources. It represents a elementary policy rule which will be referred to in a PA Policy Rule to grant access privileges to roles.

A PA Policy Rule is used to define the permission-role assignment relationship of the RBAC model. However, different from the standard RBAC model, our PA Policy Rule can specify conditional permission assignments to roles, which is a very useful feature for system administration (e.g., backup operations are only allowed after midnight). Another feature is that based on the policy information model which will be introduced later, when the PA Policy Rules are parsed senior roles will inherit permissions from junior roles automatically.

A UA Policy Rule is used to define the user-role assignment relationship of the RBAC model. It specifies the conditional connections between roles and selected subjects in a session. The constraints on conflicting roles during the assignment are also defined in this Policy Rule to achieve separation of duty.

The mapping relationship between the functionality layout of our new types of Policy Rules and the RBAC Model facilitates a policy author to design access control policies according to a well-known security model. It is worth noting that we did not design any types of Policy Rules for session management, user and role activation and role administration, because they are closely related to specific implementation issues of a RBAC deployment system. These are outside the scope of a policy language. But some of

these implementation issues in the WBEM infrastructure are discussed in section 5.

Two design features helped us realized our second design objective: compliance with CIM standards. First, our CIM-SPL RBAC extension fully inherited the original language features designed for CIM, such as the import statement, the built-in "collection" operators, etc. Second, we extend the policy information model in the CIM Policy Schema for describing our policies in a way that follows the CIM specification, as shown in Figure 5. For simplicity the prefix CIM_ has been removed from the names of classes. The gray classes and related associations are introduced to describe the policy components in our CIM-SPL extension. The others are defined by basic CIM Schemas.

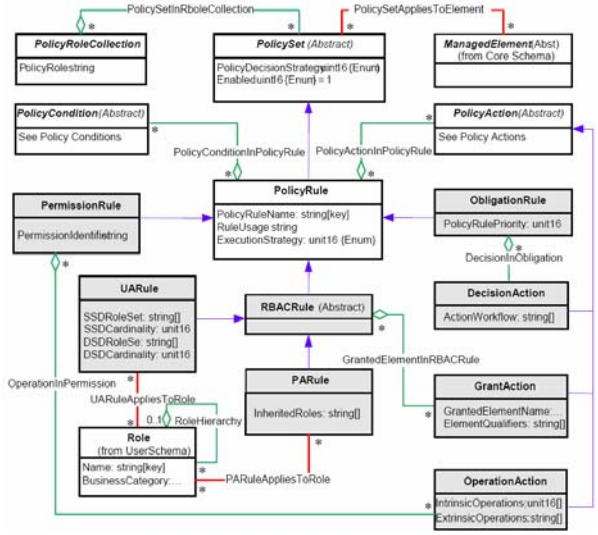

Figure 5. Extended CIM Policy Model for CIM-SPL RBAC Extension

The advantages of this design are twofold. First, it provides a unified way to define and manipulate policy components in the same way as other CIM objects. Second, it can leverage the existing information in the CIM repository by using basic model classes and associations. For example, the role class defined in the CIM User Schema is used in our extended CIM policy model to express the role information in the RBAC model. A corresponding RoleHierarchy association class is added to represent the RBAC role hierarchy model, which will be used to find the role hierarchy for a senior role and then to inherit the junior roles' permissions during the compiling process of a PA Policy Rule. Although all CIM-SPL policy rules are written in plain text, this model will provide an efficient way for a policy compiler to parse CIM-SPL RBAC policy rules. Then the encoded policy components can be further mapped into a LDAP repository according to the IETF standard [6]. It will greatly facilitate the retrieving process during policy evaluation.

An overview of the syntax of policy rules of our CIM-SPL RBAC extension given in Figure 6.

```
import <MOF Name>::<CIM Class Name>:<Condition>
strategy <Execution Strategy>
policy{
   PolicyName "policy name string";
   declaration { <constant and macros> }
   condition { <Java-like boolean expression> }
   decision { <action workflow> }
}
                    (a)
import <MOF Name>::<CIM Class Name>:<Condition>
policy {
   PolicyName "policy name string";
   declaration { <constant and macros> }
   operation{ <CIM Operation Identifiers> }
}
                    (b)
import <MOF Name>::<CIM Class Name>:<Condition>
policy {
   PolicyName "policy name string";
   declaration { <constant and macros> }
   condition { <Java-like boolean expression> }
   grant{ <Permission Identifiers>}
}
                    (c)
import <MOF Name>::<CIM Class Name>:<Condition>
policy {
   PolicyName "policy name string";
   SSOD [<role names >]:<cardinality>;
   DSOD [<role names>]:<cardinality>;
   declaration { <constant and macros> }
   condition { <Java-like boolean expression> }
   grant{ <role identifiers> }
}
                    (d)
```

Figure 6. Syntax Schema of Policy Rules in the CIM-SPL RBAC Extension

According to the third design objective, our extension follows the original language style. However the new Policy Rules do not use the original policy group structure any more, hence the corresponding execution strategy field is not needed. In all types of Policy Rules, a PolicyName element is added for policy compiling and retrieving. As shown in Figure 6(a), the rest of the Obligation Policy Rule is the same as the original policy rule.

The syntax of a Permission Rule is shown in Figure 6(b). Each Permission Rule is written against a managed resource class or selected instances by using the condition constraint in the import statement. For writing meaningful Permission policies in an easy way, predefined identifiers for intrinsic CIM operations are provided according to the names of functionality groups listed in Table 1, e.g., BasicRead. Other CIM

operation identifiers in the operation statement are specific method names of a CIM class.

The syntax of a PA Rule is shown in Figure 6(c). The permission identifiers in the grant statement are PolicyName strings defined in corresponding Permission rules. A PA rule has a condition statement that must be true when a user assign to the role wants ti exercise the permission.

The syntax of a UA Rule is show in Figure 6(d). Regarding "users" in the WBEM context, they may be people or non-human entities, such as a service running as part of an application system. So the CIM user schema factors the user into several classes. In the condition statement, instead of simply passing a user ID, our UA Policy Rule can pass a condition on attributes of various user classes. This feature is quite suitable for applications with an unknown user population, e.g., web services and user attributes like location or credentials can be used the decide whether the user is assigned to the role. The Static Separation of Duty (SSD) constraint defines the conflicting roles that can not be assigned to a user at the same time. The Dynamic Separation of Duty (DSD) constraint defines the role set that can not be simultaneously activated by a user within a session. Each constraint consists of two parts. The first part is a collection of role name strings. The second part is a cardinality integer. The role identifiers in the grant statement are name strings of role instances.

4. Examples

The first example is to demonstrate the representation capability for basic RBAC and hierarchical RBAC policy rules. Assume there is a WBEM-based computer management application used by an IT department. The junior role “monitor” in the department only wants to gather some information. For example, he wants to know the number of processes running in a managed computer with a particular operating system installation. This can be achieved by invoking an intrinsic CIM operation “GetProperty” in the BasicRead functionality group to read the value of the NumberOfProcess property of the CIM_OperatingSystem class. A senior role “operator” not only wants to gather information but also needs to do some maintenance operations, such as to set the time zone, to reboot or to shutdown the operating system. These can be realized by executing the “SetProperty” intrinsic CIM operation in the BasicWrite functionality group or invoking corresponding methods of the CIM_OperatingSystem class respectively. First we define two permission policy rules as shown in Figure 7(a) and Figure 7(b). Then two PA policy rules are defined to assign corresponding permissions to the “monitor” role and the “operator” role as shown in Figure 8(a) and Figure 8(b) respectively.

```
import CIM_X_XX_XXXX::CIM_OperatingSystem;
policy { // permission policy rule
   PolicyName ”Browse”;
   operation{ BasicRead; }
 }
                              (a)
import CIM_X_XX_XXXX:: CIM_OperatingSystem;
policy { // permission policy rule
   PolicyName “Manipulation”;
   operation{
       BasicWrite;
       reboot;
       shutdown;
    }
}
                              (b)
```

Figure 7 Two Permission Policy Rules to the CIM_OperatiingSystem class

```
import CIM_X_XX_XXXX::role:name=”monitor”;
    policy { // PA policy rule
   PolicyName ”MonitorPA”;
    condition{ }
   granted{ Browse; }
}
                              (a)
import CIM_X_XX_XXXX::role:name=”operator”;
policy { // PA policy rule
   PolicyName OperatorPA;
   condition{ }
   granted{ Manipulation; }
}
                              (b)
```

Figure 8 Two PA Policy Rules for the role “monitor” and the role “operator”

For simplicity, we did not define any conditions in the PA policy rules. Since there is a RoleHierarchy association, when the PA policy rules are parsed, all permissions of the “monitor” role will be inherited by the “operator” role. Note that a real management activity needs a set of permissions. For example, if the role “monitor” wants to know the number of processes of a running operating system on the computer named “sever1.ibm.com”, he needs to enumerate the instances of the CIM_ComputerSystem class to find this computer first, and then get the running operating system by traversing the association class CIM_RunningOS. It is obvious that a fine-grained access control on this activity can be achieved with our policy language by defining permission policy rules on various objects and granting them to appointed roles.

The second example is to demonstrate how to specify access control policy rules with separation of

duty constraints. Assume there is a WBEM-based information management application in an enterprise research department. The CIM_Proposal class is defined to represent the information of a research proposal. It has two methods. One is "submit". Another is "approve". The policies for research department are: only researchers who are full-time employees can submit a proposal; and, only a person whose title is "manager" is qualified to be a member of the committee which has the right to approve a proposal. Two permission policy rules are defined to represent the privileges to submit or to approve a proposal respectively, as shown in Figure 9(a) and Figure 9(b).

```
import CIM_X_XX_XXXX::CIM_ Proposal;
policy { // permission policy rule
   PolicyName "SubmitProposal";
   operation{ submit; }
}
                        (a)
import CIM_X_XX_XXXX::CIM_ Proposal;
policy { //permission policy rule
   PolicyName "ApproveProposal";
   operation{approve;}
}
                        (b)
```

Figure 9 Two Permission Policy Rules to the Class CIM_Proposal

Then these two permissions are assigned to the "researcher" role and the "committeeman" role respectively by two PA policy rules as shown in Figure 10(a) and Figure 10(b).

```
import CIM_X_XX_XXXX::role:name="researcher";
policy { // PA policy rule;
   PolicyName ResearcherPA;
   condition{ }
   grant{ SubmitProposal;}
}
                        (a)
import CIM_X_XX_XXXX::role:name="committeeman";
policy {// PA policy rule;
   PolicyName ResearcherPA;
   condition{ }
   grant{ ApproveProposal; }
}
                        (b)
```

Figure 10 Two PA Policy Rules for the role "researcher" and the role "committeeman"

Finally, two UA policy rules with separation of duty constraints are defined in Figure 11. In Figure 11(a), a built-in operator "isWithin" is used to express a time condition constraint to assign a user with the "researcher" role to submit a proposal in a limited time period. The "researcher" role belongs to the SSD role set with cardinality 1, so users can not be assigned more than one role from this set. Similarly, the "researcher" role also belongs to the DSD role set with cardinality 2, and hence authorized users can activate no more than two roles at the same time in a session. Similar contraints are also defined in Figure 11(b).

```
import CIM_X_XX_XXXX::person:EmployeeType="FullTime";
policy { // UA policy rule
   PolicyName "ResearcherRA";
   SSOD ["Researcher", "Committeeman"]:1;
   DSOD ["Researcher", "staff", "CEO"]:2;
   declare{
      CurrentDay=getDayofYear(time);
   }
   condition{
    isWithin(CurrentDay, 2008-2-1 TZ = Newyork,
              2008-07-31 TZ = Newyork);
   }
   grant{ researcher; }
}
                        (a)
import CIM_X_XX_XXXX::person:Title="manager";
policy { // UA policy rule
   PolicyName "CommitteemanRA";
   SSOD ["Researcher", "Committeeman"]:1;
   DSOD ["Committeeman", "Engineer", "CEO"]:2;
   grant{  committeeman; }
}
                        (b)
```

Figure 11 Two UA Policy Rules with Separation of Duty Constraints

5. An Implementation Framework for OpenPegasus

One of the design principles of both the original CIM-SPL and our extension is to avoid having specific implementation concerns intertwined with language characheristics [7]. It provided the maximum flexibility to deploy a policy-based system. However, we propose an open framwork of our CIM-SPL RBAC extension in the OpenPegasus WBEM infrastructure to demonstrate its usability, as illustrated in Figure 12. Our implementation framework follows the Policy Enforcement Point/Policy Decision Point (PEP/PDP) approach proposed by IETF [8]. In our framework, the PEP is a modified OpenPegasus CIM Sever. It interprets all CIM operation requests coming from the WBEM clients, and then sends policy requests to the PDP. The PDP, our CIM-SPL RBAC policy sever, is implemented basically as a method provider in OpenPegasus. So the communication between the PEP and the PDP uses the standard provider interface APIs of OpenPegasus. The CIM-SPL RBAC provider has an internal state database for storing state information for policy requests. These states are used to evaluate a policy rule with complex constraints, e.g., to evaluate a

DSD contraint in a UA policy rule one needs to search informaion among active sessions. The Policy Repository is a LDAP sever that stores the RBAC policy component objects described in Section 3. The policy management tools are used to edit and compile policies into the repository, and aslo act as a interface for administering the PDP service.

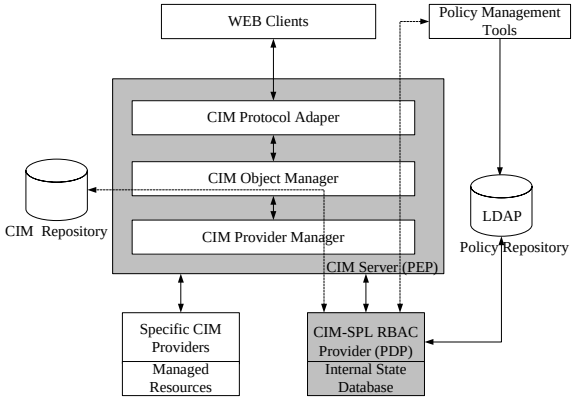

Figure 12 An Implementation Framework of CIM-SPL RBAC Extension

Due to space limitations, we only discuss some main implementation issues and choices in the following.

A. Compatibility with the Namespace-based Access Control Mechanism in OpenPegasus

The current OpenPegasus implementaion has a Namespace-based access control mechanism. All CIM operations are divided into two groups of READ and WRITE in the OpenPegasus implementation. An interface named "cimauth" is provided to define authorization rules to grant the READ or WRITE permission on a target Namespace to users. Since the Name-based access control is enforced before the CIM server delivers the CIM operation to a specific provider in which the managed resource classes are realized, the RBAC mechanism in our framework can be cascaded to it to achieve a fine-graind access control. Moreover, the Namespace-based authorization rules can be easily represented by our CIM-SPL RBAC extansion language, although the literal meaning of a permission policy rule defined against a Namespace means the approval of operations on the Namespace class, an appropriate PEP implementation in the CIM sever who sends the policy request should know from the policy evaluation whether the CIM operation is allowed to be further processed to a specific class within this Namespace. This provides the possibility to integrate the Namespace-based access control into our framework in the future.

B. PEP Implementation

The internal components of the CIM server are illustrated in Figure 12. There are many places that can be chosen to implement the RBAC PEP. The Provider Manager is a component responsible for the routing of requests and responses between the CIM Object Manager and the CIM Providers. So we choose the provider manager to implement the RBAC PEP in our framework. Another advantage of this design choice is that the Provider manager is also responsible for the maintainence of the Provider registration, and Provider loading and unloading, so it is possible to realize a policy registration mechanism for each provider. It provides the possibility to deploy a policy management system with distributed administration. Note that the Namespace-based authorization process is implemented in the CIM Object Manger. As discussed above, it can be cascaded to our framework.

C. Data Gathering

Unlike certain policy languages that assume a model of how the data required for policy evaluation will be gathered, the specification of CIM-SPL and our RBAC extension is completely independent of how the required data is gathered. The data gathering mechanism depends on the policy server implementation. Based on the OpenPegasus implementation, the CIM-SPL RBAC PDP server gathers the required data for policy evaluation through three approachs. First, it retrieves predefined constant data from the policy repository. Second, it collects state data from the internal database. Third, a "connectlocal" connection is established between the PDP sever and the CIM sever to gather data from the CIM repository. The "connectlocal" is an interface provided by the OpenPegasus implementation. With this connection the PDP server can act as a CIM client to gather data from the CIM repository through CIM operations.

D. Additional Services

One of our design decisions is not to specify the related session and role management services. It is envisioned that when a policy-based RBAC application system is deployed, as a part of the deployment, these services will be specified in the implementation. In fact, a set of SSL-based authentication and session management services are already provided by the OpenPegasus implementation. A user from a WBEM client is activated and authenticated in a SSL session before it can perform an action. As to the role management, basic properties and methods for role management in a managed system have been specified in the CIM Role Based Authorization Profile. Some common services, such as CreatRole(), DeleteRole() and ModifyRole(), etc., are

defined in the CIM_RoleBasedAuthorizationService class. These serivces can be seamlessly integrated with our framework to provide a complete solution for deploying a WBEM-based RBAC management system.

6. Related Works

Except for CIM-SPL, CQL (CIM Query Language) [9] is the only suggested policy language in the WBEM infrastructure. However, CQL assumes a relational model. This imposes a complicated syntax to represent policies for CIM constructs. Moreover, CQL can not specify access control policies directly. In the CIM context, a close work to ours is decribed in [10]. They presented a framework based on PCIM [11] for storing and enforcing RBAC policies in distributed heterogeneous systems. But they did not address the access control problem in the WBEM infrastructure and their RBAC policies are represented only by using CIM constructs instead of a formal policy language as in our work. Ponder [12] is an object-oriented general policy language for specifying access control rules called authorization rules as well as general purpose management rules. Ponder uses the role concept to provide a semantic grouping of policies with a common subject. There is no direct relationship between the role-based Ponder authorization rules and the RBAC model. In the Ponder deployment framework, policy management operations are carried out by invoking methods of the policy object through enforcement agents. So it is hard to combine this model with the WBEM infrastructure. Another important work on access control policy languages is the XACML policy language[13] which aslo adopts the PDP/PEP deployment framework. The specification for RBAC policy rules are provided in the XACML RBAC Profile [14]. But it can not be used to define seperation of duty contraints and it only supports a role assignment specification based on subject IDs.

7. Conclusion

This paper has presented a CIM-SPL policy language extension for Role-based Access Control. The CIM-SPL RBAC extension inherited all characteristics of the original CIM-SPL policy language and it fully complies with the WBEM standards. With this CIM-SPL extension, access control policies can be easily defined for managed resources stored in CIM constructs according to the standard NIST RBAC model. Although the language characteristic makes it flexible to be implemented by various software patterns, an implementation framework in the OpenPegasus WBEM infrastructure is proposed to demonstrate the usability of our CIM-SPL RBAC extension. However, the ultimate validation of a policy language will be demonstrated by the degree of its adoption. So, one of our ongoing works is to realize a complete open policy framework with CIM-SPL in the OpenPegasus open source WBEM implementation. We expect it will be helpful to promote more and more WBEM-based management applications adopting the policy technology with CIM-SPL in a standard open architecture.

8. References

[1] The Open Group: OpenPagasus: C++ CIM/WBEM manageability service broker. http://www.openpegasus.org

[2] Raouf Boutaba, Issam Aib, "Policy-based Management: A Historical Perspective", Journal of Network and Systems Management. Volume 15, Number 4 / December, 2007, pp. 447-480.

[3] DMTF, "CIM Simplified Policy Language (CIM-SPL). Specification DSP0231", v1.0.0a, 10 Jan 2007.

[4] DMTF, "Specification for CIM Operations over HTTP", Version 1.2, 2004.

[5] Ferraiolo et al., "The NIST Model for Role-Based Access Control: Towards a Unified Standard", ACM Trans. Information and System Security, vol. 4, no. 3, Aug. 2001, pp. 224-274.

[6] J. Strassner, E. Ellesson, B. Moore, R. Moats, "Policy Core Lightweight Directory Access Protocol (LDAP) Schema", IETF RFC 3707, Feb. 2004.

[7] Dakshi Agrawal, Seraphin B. Calo, Kang-Won Lee, Jorge Lobo, "Issues in Designing a Policy Language for Distributed Management of IT Infrastructures", 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007, pp. 30-39.

[8] Yavatkar, R., Pendarakis, D., Guerin, R., A framework for policy-based admission control, IETF Network Working Group, RFC 2753, January 2000

[9] DMTF, "CIM Query Language Specification". Version 1.0.0h edn. (2006)

[10] Timothy E. Squair, Edgard Jamhour, Ricardo C. Nabhen, "A RBAC-Based Policy Information Base", Sixth IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 171 – 180.

[11] Moore, B., Ellesson, E., Strassner, J., Westerinen, A, "Policy core information model (PCIM) version 1 specification" Request for Comment 3060, 2001, Network Working Group.

[12] Damianou, N., Dulay, N., Lupu, E., Sloman, M., "The ponder policy specification language". Proceedings of the International Workshop on Policies for Distributed Systems and Networks, 2001, pp. 18–38.

[13] OASIS, "eXtensible Access Control Markup Language (XACML)", version 1.03. OASIS Standard, Feb. 2003.

[14] Anderson, ed., "XACML Profile for Role-Based Access Control (RBAC)", OASIS Access Control TC committee 01, 13 Feb. 2004.