

Control access policies for distributed resources

BEN MBARKA MOEZ
(benmbark@enseirb.fr)

January 16, 2007

Abstract

Policies are more and more used to manage access control for distributed network applications . This interest essentially stems from the dynamicity that provides this approach. Policy-based management regards the multiple resources composing the system as a single abstract resource. This leads to write high level policies which need to be refined before to be distributed to the concretes resources. As, these resources should work as a global system, internal coordination may be needed. This paper proposes a model and a refinement process able to automatically produce low level policies. It also discuss a conceptual coordination model and suggests implementation issues.

1 Introduction

Modern information systems and network applications are more and more complicated and usually use distributed resources to offer a global service. These applications usually have rules and constraints to deny or grant access requests. Classic solutions[5] [6] and access control architectures don't make separation between the system and rules governing it. This approach, usually leads to recode some functionalities or to restart the system if the rules governing it are updated. Today, more information systems are dynamic (in terms of size and access control rules) and need dynamic management. Such management should allow the administrators to update access rules without updating the code of any program or restarting the system. A recent approach is to use policy-based management to control access to multiple distributed resources. A control access policy can be defined by[7] : Set of rules defining the behavior of a system to manage access requests. This approach makes distinction between the system and rules composing the policy. This policy is not part of the system itself but the latter should be able to interpret it. Policy-based management provides abstraction of the concrete resources. Administrators can write high level policies by making abstraction of equipment specific configuration. These policies need to be refined (or decomposed) to be specific to the resource where they should be applied. This process is called "policy refinement process"[2].

Policies data are stored in servers known as "policy decision points" (PDPs). Routers and switches, known as "policy enforcement points" (PEPs), query the PDPs for the required information. Policy-based systems can be build by using one or more PDP. For many network applications, distributed resources may be managed by different administrators which leads to have distributed PDP. Moreover, replacing a centralized PDP with a local one associated to each resource or set of resources can increase the performance of the application. Indeed, the use of a centralized PDP with high level policies is a bottleneck to performance because every access request needs to be managed by the same centralized PDP. However, the distributed approach lacks of the ability to coordinate access control decisions.

This paper is structured as follows. Section 2 introduces a conceptual model for representing resources and policies and proposes through an example the refinement process. Section 3 introduces a conceptual model for a coordination object and coordination policies and discuss some implementation issues.

2 Automated decomposition of access control policy

The administrator can focus on the global behavior of the system by writing high level policies. However, these policies are not specific to the equipment (hardware, vendor...) and to the concrete resources attributes. We present here a method to automatically refine a high level policy to low level one[2] with less components and more specific to the concrete resource. First, we introduce models for representing resources and policies.

2.1 The resource type hierarchy

Multiple resources can be represented by a hierarchical structure. A directed graph (DAOG[2] : Directed AND/OR Graph) can be used to represent this structure (figure 1). The most abstract resource is an origin node (e.g information system). Internal nodes are also abstract resources (e.g cluster). Concrete resources are the leaf nodes (e.g printer). Each resource can be described by a set of attributes (figure 1). These can be subject, resource, action or environment attributes. *Action* and *type* attributes are mandatory for each node. The edges traduce containment relationship between the resources. We say resource S2 is contained into S1 if there is a path from the node S1 down to the node S2. Accordingly, we say type (or action) associated to S1 is a containing type (or action) of the type (or action) associated to S2. Figure 1 gives an example : *node1* is contained into *cluster*, so the type *cluster* is a containing type of the type *node1* and the action “run” is a containing action of the action “compute”.

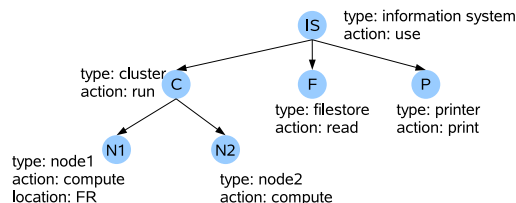


Figure 1: DAOG example.

2.2 Policy specifications

An access control policy is a set of rules able to decide to grant or deny access requests. Policies can be represented with arithmetic and logical expressions[2]. For example to describe “only administrators can read from the filestore”, the policy may be : $type(R) = filestore \text{ AND } action(R) = read \text{ AND } role(S) = administrator$. The component $type(R) = X \text{ AND } action(R) = Y$ is mandatory for any policy. Y denotes a specific action supported by the resource X . Each component of the logical expression may evaluate to TRUE or FALSE at run time. The PDP grants an access request only if the hole expression is evaluated to TRUE.

2.3 Policy refinement process

The aim of the policy refinement process[2][3] is to automatically produce from an abstract high level policy a low level policies for concrete resources. The produced policies will be as specific as possible to the resource where they should be applied. Given any node in a DAOG, its policy can be produced by the following steps. We will apply these steps with the DAOG given at the figure 1 and as high level policy we take $P : type(R) = IS \text{ AND } action(R) = use \text{ AND } role(S) = student \text{ AND } location = FR \text{ AND } day = Monday \text{ AND } \neg(action(R) = print \text{ AND } type(R) = printer)$ which means that a student can use the information system(IS) at the location FR on Monday but can not print using the printer. The aim is to produce a low level policy for the node $N1$.

The first step is to make the attributes *type* and *action* of P specific to the node $N1$ when it is possible. Using containment relationship represented into the DAOG, we replace type or action values associated with a node M into P by type or action value associated with the resource N if the resource N is contained into the resource M . After this step the first intermediate policy is $P_1 : \mathbf{type(R)=N1 \text{ AND } action(R)=compute} \text{ AND } role(S) = student \text{ AND } location = FR \text{ AND } day = Monday \text{ AND } \neg(action(R) = print \text{ AND } type(R) = printer)$. The containment relationship implies compatibility relationship between resource actions. We replaced *IS* with $N1$ and *use* with *compute* because the former type (or action) is a containing type (or action) of the latter type (or action).

The next step is to evaluate all relational expressions not involving action or type attributes. Any relational expression may be evaluated to TRUE, FALSE or INDETERMINATE. In the latter case we should keep the expression because it may take value at run time. Into P_1 , $location = FR$ evaluates to TRUE at the node $N1$. However, $day = Monday$ and $role(S) = student$ evaluate to INDETERMINATE at refinement time but will take values from the request context at run time. So, the new intermediate policy P_2 will be : $type(R) = N1 \text{ AND } action(R) = compute \text{ AND } role(R) = student \text{ AND } \mathbf{TRUE} \text{ AND } day = Monday \text{ AND } \neg(action(R) = print \text{ AND } type(R) = printer)$.

Now to make P_2 more specific to the resource $N1$ we replace relational expressions involving type or action attributes that will always evaluate to FALSE at run time with FALSE. For example : $action(R) = print$ and $type(R) = printer$ will always evaluate to FALSE at the node $N1$. Therefore, the new policy P_3 is : $type(R) = N1 \text{ AND } action(R) = compute \text{ AND } role(R) = student \text{ AND } \mathbf{TRUE} \text{ AND } day = Monday \text{ AND } \neg(\mathbf{FALSE \text{ AND } FALSE})$.

The last step consists at simplifying the last intermediate policy P_3 to finally have the low level policy P_{N1} of the the node $N1$: $type(R) = N1 \text{ AND } action(R) = compute \text{ AND } role(R) = student \text{ AND } day = Monday$.

The produced policy is specific to the node $N1$ and is simpler than the first one in terms of the number of components. To have the low level policy for a resource instance we need to apply the same steps to the produced policy. The process ensures that at any level the intermediate policy is correct. It means that if it returns the value TRUE according to a run time context, the previous policies return the same value according to the same run time context.

3 Coordination

To increase the performance of the distributed system, each resource can have its own PDP which uses the local policy to decide about access requests. However, this approach lacks of the ability to coordinate access control decisions. For example, if we have the policy "every user can withdraw at maximum 100\$ per day", without coordination between withdrawal points, any user can withdraw 100\$ from each withdrawal point because locally the policy constraint is respected although it is obviously broken from a distributed perspective. Furthermore, even with a centralized PDP,

coordination may be needed over time. Indeed, a PDP is supposed to be stateless. It means that an access control decision is made according to the current policy and in isolation to all past, present and future decisions. Therefore, if any access control decision will produce state changes in the request context and these changes will affect future access control decisions, then coordination is required[1]. We will describe here a conceptual model to allow centralized or distributed PDPs to coordinate their decisions.

3.1 Coordination object

It is composed of subject, resource, and environment attributes. Conceptually, the coordination object defines the attributes needed to ensure the coordination between all access control decisions in a distributed system. Examples are given in the next paragraph.

3.2 Coordination policies

In order to allow coordination, we can add coordination attributes to the policy. For example, a policy to limit memory usage to 1Gb, without coordination may be like : $type(R) = memory \text{ AND } action(R) = use \text{ AND } amount < 1$. This policy is not enough if we want to ensure the limit over time (The total amount of multiple uses must not exceed 1GB) or if the PDPs with this local policy are distributed to many memory resources. Therefore, we need to add coordination attributes to the policy. Needed attributes are known from the request context attributes which need coordination. In this example we need to coordinate the requested amount of memory. So the coordination attribute can be the *storage* already used. The policy becomes : $type(R) = memory \text{ AND } action(R) = use \text{ AND } amount + storage\{userID(S)\}(C) < 1$. Where *storage* is a coordination attribute of the coordination object *C*. *userID(S)* is a subject parameter of this attribute. It means that *storage* has different value for each subject defined by its *userID*. At run time, coordination attributes are evaluated before decision making.

In opposition to resource or environment attributes, coordination attributes may need to be updated. For example, for the last policy we need to increment $storage\{userID(S)\}$ by the value of the current amount if the request was granted. So, for the next request we will have the account of all previous uses. An obligation element[1][9] can be added to the coordination policy to define how and when coordination attributes must be updated. Currently, an obligation element can be described by the following[1] : $@\{< Chronicle = [After - Request|Before - request], \{V_1 \leftarrow E_1, \dots, V_k \leftarrow E_k\} >\}$. Where *After - Request* means updates must be done after the request is enforced, while *Before - request* means before. $\{V_1 \leftarrow E_1, \dots, V_k \leftarrow E_k\}$ is a set of assignments. For the last policy example, the obligation element may be : $@\{< Chronicle = After - Request, \{storage\{userID(S)\} \leftarrow storage\{userID(S)\} + amount\} >\}$.

We have already shown how to refine a high level policy to a low concrete one. When coordination is needed a coordination policy and obligation elements may be included to the high level policy before the refinement process. In the case where the coordination policy does not have action or type attributes, no refinement is possible because the coordination constraints apply to all resources. Indeed, except action and type attributes, coordination attributes always evaluate to *INDETERMINATE* at the refinement time because their semantic is only available into the PDP at run time.

4 Implementation

We will discuss storage, data accessing and distribution issues.

4.1 Storage

To efficiently store coordination data, we need high performances for read and write operations with multiple concurrent access support. Coordination attributes may be stored into a data base which offers all these benefits. Each attribute can be stored into one table. The dimensions of the tables may be calculated from the dimensions of the coordination object attributes. An attribute has multiple optional dimensions [SubDim, ResDim, ActDim, EnvDim] where SubDim, ResDim, ActDim, EnvDim denote respectively subject, resource, action and environment dimensions. For example, $amount[\{userID(S)\}]$ has one subject dimension, so the coordination attribute is a 1-D attribute. $amount[\{userID(S)\}]$ means that we have to store an amount value for each user (subject) identified by its $userID$. $usage[\{userID(S)\}, \{date(E)\}]$ has, in addition to one subject dimension, one environment dimension. It means that we have to store a *storage* value for any combination of $userID$ and the environment context identified the $date$. The dimension of an attribute may be defined by : $|SubDim| + |ResDim| + |ActDim| + |EnvDim|$. So the attribute $usage$ is a 2-D attribute. The coordination attribute $amount[\{userID(S)\}]$ can be stored into the table $amount$ with 2 columns: one for the $userID$ value and one more for the amount value. The attribute $usage[\{userID(S)\}, \{date(E)\}]$ needs a 3-columns table: one for each attribute dimension plus one for the attribute value. In general case, a coordination object may be stored into a data base with a table for each coordination attribute. The dimension of each table is the dimension of the associated attribute plus 1.

4.2 Data accessing

At run time, we need to retrieve attribute values from the data base. This task is done by the PIP (Policy Information Point)[8] which is a PEP component. Hence the PIP has the attribute name with its parameter values, it can retrieve the attribute value using a standard SQL request (e.g *SELECT value FROM amount WHERE userID = 1*). If no record has been already stored for the $userID$ 1 then a new line is inserted with the initialization values specified into the policy. Thus, the coordination object can be built dynamically. At initialization time, we just need to know the coordination attribute names to build the empty tables. Any table can be updated at run time using obligation elements. Retrieval and update (before or after the request depending on the chronicle) should be implemented as an atomic operation to ensure consistency of coordination operations.

4.3 Coordination object distribution

Initially the coordination object is defined by a set of empty tables. These tables may be centralized, partitioned and then distributed, or replicated. For more performance, it is suggested that tables containing resource type attributes are partitioned horizontally and distributed to the sites where each resource resides. Moreover, for each part we can delete the resource type column because it will has the same value at all the run time. The remaining tables are centralized and should be accessible for all distributed resources.

Conclusion

Policy-based management provides an appropriate and dynamic way to manage access control for distributed applications. The paper proposed a conceptual model and method to refine high level policies. The system administrator can make abstraction of the equipment configuration details and write high level policies. The refinement process produces automatically low level policies which can then be distributed to concrete resource instances. This process is only based on removing

redundant and unused components of the policy at the level of the concrete resource and may be optimized. Modern applications are more and more complicated and the access control decisions need to be taken as faster as possible. Distribution of policies allows to avoid the centralization of requests which is a bottleneck for the performance. The second part of this paper, presented coordination issues for distributed but also centralized PDPs . The proposed conceptual model lacks yet of how to specify the initialization values for the coordination attributes and how to handle attributes with multiple values.

In addition to refinement and coordination problems, research into policy-based management has focused into policy specification languages , deployment architectures and communication protocols between the different components(PDP, PEP, PIP...) .

References

- [1] L. Su, D. W. Chadwick, A. Basden and J. A. Cunningham, "Coordination between Distributed PDPs"
- [2] L. Su, D. W. Chadwick, A. Basden and J. A. Cunningham, "Automated Decomposition of Access Control Policies"
- [3] A.K Bandara, E.C. Lupu, J.Moffett and A.Russo, "A Goal-based Approach to Policy Refinement", Proceeding of the fifth IEEE International Workshop on Policies for Distributed Systems and Networks(POLICY'04/)
- [4] DamianouN, Bandara A.K., Sloman M., Lupu E., "A survey of policy specification approches", <http://www.doc.ic.ac.uk/~mss/Papers/PolicySurvey.pdf>, 2002
- [5] ISO, "Management Framework". ISO 7498-4, 1989
- [6] ITU-T, "Principles for a telecommunication management network", ITU-T M3010, 1996
- [7] Sloman M., "Policy Driven Management For Distributed Systems", Journal of Network and Systems Management, vol 2, no. 4, Dec. 1994
- [8] eXtensible Access Control Markup 3 Language (XACML) Version 1.0, <http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf>
- [9] Moore B., "Policy Core Information Model(PCIM) extensions", RFC 3460, IETF, 2003