Role-Based Resource Management

Evangelos Kotsovinos and Tim Harris
University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, UK, CB3 0FD
{firstname.lastname}@cl.cam.ac.uk

Abstract— **In this paper, we propose a flexible resource management system suitable for expressing and applying policies to apportion resources in diverse and heterogeneous global-scale public computing systems. We do this by devising a role-based system, which allows expressing complex role membership and resource allocation policies.**

I. Introduction

Expressing and applying resource allocation policies in federated distributed systems is a hard task; resources are controlled by a large number of individual physical and administrative entities. There is a need to express policies defined by the administrators of each resource as well as by central "authorities", and combine the policies in a reliable and well-defined manner.

This need is becoming increasingly pressing in settings such as utility and public computing infrastructures [6], overlay testbeds [1] and computational grids [8]. The problem of resource management in federated systems is becoming an important one.

To allow representing, applying, and combining complex resource usage policies originating from a large variety of sources, we propose a *role-based resource management* scheme. The owner of the resources and other stakeholders can specify the desired constraints and policies, expressing which users or groups of users can be allowed access to which parts of the resources.

Our approach can be seen as a development of Role-Based Access Control (RBAC), which has received significant research focus in the past. In RBAC, "access control decisions are determined by the roles individual users take on as part of an organization". The basis of RBAC is the concept of a *role*, which is essentialy a user grouping mechanism. RBAC allows the administrator to specify which users should enter which role, as well as which roles should be granted access to which resources. The nature of RBAC decisions is binary; one can either be granted access or not. Thus, when role entry conditions are conflicting, the most common approach is that if there is one that denies access then it simply overrides the others, in accordance to the "least privilege" principle [13].

When considering applying the same concepts on distributed resource management, a key technical difference that emerges is that, in contrast to access control, which is binary, resource management is *quantitative*; the question then becomes how much access to grant a user to a resource, rather than simply whether to grant access or not. Moreover, in global-scale systems there is usually no notion of a central authority controlling the distributed resources. Therefore, one can expect that – possibly overlapping or conflicting – policies and roles will need to be defined by a number of heterogeneous entities, that coexist under separate administrations.

A problem that emerges as a result of these challenges is *conflict resolution*. Imagine, for exposition, a case where Bob, the resource owner, allocates 10% of its resources to Computer Science students and 8% to IEEE members, and user Alice belongs to both roles. Then, would Alice be given 10%, 8% or 18%? Is it Alice that Bob wanted to allocate 8% to, or is it all IEEE members on aggregate? And what happens if all 8% is already allocated to IEEE members when Alice's request appears?

It is easy to see that there is a need for a flexible, expressive and comprehensible system, able to combine role entry and resource allocation constraints, to allow efficient resource management in global-scale public computing infrastructures. For the required resource management system to be able to operate successfully in a distributed, global-scale setting, we need to make sure it is designed and implemented in a decentralized fashion. There is no notion of a central authority responsible for the way resources can be managed, and to make our system efficient it is necessary to avoid bottlenecks and single points of failure. The *scalability* of our system can clearly only be served well by distribution of computation.

The rest of this paper is structured as follows. In Section 2 we describe the Role Description Language that is used. In Section 3 we analyze the proposed architecture, and in Section 4 we conclude.

II. Policy Description

The main elements of our Role-Based Resource Management architecture are the following. There is a number of *users*, who request *resources* from a number of entities. Those entities can declare *roles* that define the groups that users can enter as well as the conditions of entry. To define how resources are to be apportioned between roles, entities can specify *constraints*.

Below, we explain how roles and constraints can be defined and combined to express complex resource allocation policies in a flexible, efficient and comprehensible manner.

A. Roles

A *role* is essentially a grouping mechanism for users. There are a number of actions to be taken to define the roles in which users may participate, specify which users are eligible to enter which roles, and associate these roles with resource management policies.

Roles have to be *declared*, in the same sense that variables would be declared in a programming language, along with the parameters that may apply. A role declaration includes the name of the entity that declares the role, the name of the role and its parameters. There is a different name space for each entity that declares roles.

The *entry conditions* for each role need to be defined. Entry conditions specify what conditions a user has to meet to be allowed to enter the role, and how roles interact. When a client enters a role, he remains a member of the role until membership is explicitly or implicitly revoked.

The format and usage of role declarations and role entry conditions are described in more detail in the following sections.

Role naming. In an open, large-scale system, it would be difficult to enforce a single system-wide name space for roles. Instead, the approach that we take is to name roles *hierarchically*, so that each entity that defines roles can have its own role name space. Role names are of the form:

```
Declarer:Name
```

where the `Declarer` is the entity, according to which the role is defined.

Role declarations. To define a role and its parameters, a *role declaration* has to be performed. Roles can have parameters, in order to avoid having to declare different roles for slightly different occurrences of the same role. Role declaration commands are of the form:

```
Role(Parameter1, Parameter2, ...);
```

For example, if Sam wishes to define a role for users that reside in the UK, one for those who live in a particular city in the UK, and one for engineers,

```
Sam:InUK();
Sam:InUKCity(City);
Sam:Engineer();
```

Also, suppose Alice defines a role for engineers that live in Cambridge, UK, and another one for engineers of Oxford and Cambridge:

```
Alice:CamEngineers();
Alice:OxbridgeEngineers();
```

Role entry conditions. To specify which users can enter a role, one or more *role entry conditions* can be defined. Again, for flexibility and openness, role membership is designed to be *subjective*. A user is not a member of a role globally, but a member of a role according to an `Elector`.

Criteria for entering a role can be either physical properties or membership of other roles. Role entry conditions can be of the form:

```
roleEntry(Elector→Role(Parameters),
          Attribute1, Range1,
          Attribute2, Range2,
          ...);
```

in which case users whose property indicated by `Attribute1` has a value within `Range1`, and property indicated by `Attribute2` has a value within `Range2`, and so on, can enter the `Role`, according to the `Elector`. For example,

```
roleEntry(Bob→Sam:InUK(), "Country", "UK");

roleEntry(Bob→Sam:InUKCity("Cambridge"),
          "City", "Cambridge",
          "Country", "UK");


roleEntry(Bob→Sam:InUKCity("Oxford"),
          "City", "Oxford",
          "Country", "UK");

roleEntry(Jerry→Sam:Engineer(), "Occupation", "Engineer");
roleEntry(Bob→Sam:Engineer(), "Occupation", "Engineer,CompSci");
```

Alternatively, to allow clients to enter `Role` based on prior role membership of `Role1, Role2, ..., RoleN` according to the electors `Elector1, ..., ElectorN` respectively,

```
roleEntry(Elector→Role(Parameters),
          Elector1→Role1(Parameters),
          Elector2→Role2(Parameters),
          ...,
          ElectorN→RoleN(Parameters));
```

The following example shows how entry to the `CamEngineers` role, according to and as defined by Alice, can be determined. There are two conditions; the user needs to be a member of the `InUKCity("Cambridge")` role as defined by Sam and according to Bob. Also, the user needs to participate in the `Engineer` role as defined by Sam and according to Jerry.

```
roleEntry(Alice→Alice:CamEngineers,
          Bob→Sam:InUKCity("Cambridge"),
          Jerry→Sam:Engineer)
```

To allow entry to the `OxbridgeEngineers` role, according to Alice and as defined by Alice, the following conditions are specified. Users need to be engineers according to Jerry and as defined by Sam, and reside in Cambridge or Oxford according to Bob and as defined by Sam. These entry conditions can be used to make `OxbridgeEngineers` contain the union of the two roles:

```
roleEntry(Alice→Alice:OxbridgeEngineers,
          Bob→Sam:InUKCity("Cambridge"),
          Jerry→Sam:Engineer)


roleEntry(Alice→Alice:OxbridgeEngineers,
          Bob→Sam:InUKCity("Oxford"),
          Jerry→Sam:Engineer)
```

B. Constraints

To express a reservation or usage limitation on a resource, *constraint definitions* are used. A constraint definition is associated with a role. Thus, the constraint is applicable to all members of the role.

As constraints can be defined by different entities and can conflict with other constraints, for instance in cases where a user is a member of two roles for which there are two different constraint definitions for the same resource, we need an explicit way to prioritize constraints and resolve conflicts. This can be done by defining *constraint relationships*.

Below, we describe the format and usage of constraint definitions and constraint relationships.

Constraint definitions. A *constraint definition* limits or guarantees the amount of a resource that members of a role can get. Constraint definitions are of the form:

```
Elector→Role(Parameters)
       Constrainer:ConstraintKind(Resource, Parameters);
```

where `Constrainer` is the entity that is imposing the constraint, and `Role` is the role whose members – according to the `Elector` – are subject to the constraint. `ConstraintKind` is an identifier that describes what kind of limitation or reservation the constraint is meant to indicate, such as `limitEach, limitGroup, reserveEach or reserveGroup`. `Resource` is the kind of resource that the constraint applies to, and `Parameters` indicate the extent of the limitation or reservation. For example:

```
Alice→Alice:CamEngineers() LocalServer:limitGroup(CPU, 8%);
```

would limit the aggregate CPU usage on the local server by cambridge engineers – according to and as defined by Alice – to 8%.

Constraint relationships. In order to allow defining how conflicting constraints are resolved, we introduce *constraint relationships*. Each relationship gives a series of pattern-matches for existing constraints, and then a replacement constraint to be generated in their place. The format of constraint relationships is:

```
ConstraintRelationship(Constraint1,
                       Constraint2,
                       ...  ,
                       Expression1,
                       Expression2,
                       ...  ,
                       Replacement);
```

For example, to express that where different entities give constraints about maximum CPU usage, the minimum ought be taken, one can use:

```
ConstraintRelationship(
    "*→R1 X:limitEach(CPU, A)",
    "*→R2 Y:limitEach(CPU, B)",
    "X != Y",
    "*→R1 LocalServer:limitEach(CPU, min(A,B))");
```

Notice that, as the two conflicting constraints are replaced by the new one, which of the two roles the new one refers to does not make any difference, as for the conflict to exist the user must be a member of both roles. To express that, when a role is a sub-role of another one, constraints imposed by the more specific role override the ones imposed by the less specific role,

```
ConstraintRelationship(
    "*→R1 *:limitEach(CPU, A)",
    "*→R2 *:limitEach(CPU, B)",
    "R1 < R2",
    "Alice→R1 LocalServer:limitEach(CPU, A)");

ConstraintRelationship(
    "*→R1 *:limitEach(CPU, A)",
    "*→R2 *:limitEach(CPU, B)",
    "R2 < R1",
    "Alice→R2 LocalServer:limitEach(CPU, B)");
```

III. Resource allocation

In the previous sections we have described the Role Description Language used in our system, and explained how roles and constraints can be defined and managed. This section examines how our system uses the set of roles and constraints defined, in order to determine whether to grant or deny a resource allocation request.

A. Role entry

The first step to reach an admission control decision is to decide which roles a user is a member of. This process requires as input the credentials and properties of the user as well as the role declarations and entry conditins.

Entry conditions have to be examined and checked against the user's properties and credentials for the role memberships to be determined. For instance, the `"Country"` property of the user will determine if he should be allowed to enter the `"inUK"` role.

Membership of roles that have more complex entry conditions, such as prior multiple role membership, can be decided by starting checking membership from the simplest roles and going up to the more complex ones. If the user is found not to be a member of one of the roles, the process stops and the user does not enter the complex role. While the administrator has to make sure that no loops exist in role entry conditions, the development of an algorithm that produces a consistent set of role memberships is straightforward, as there are no role entry conditions that allow membership in a role on the condition that there is no membership to another one.

The output of this process is the set of *role memberships* in which the user participates.

B. Constraint processing

One of the purposes of the proposed Role-Based Resource Management system is to allow the server administrator to express policies like "Allow Cambridge Engineers access to at most 10% of the CPU, and guarantee Oxford Engineers access to 5% of the network bandwidth". To

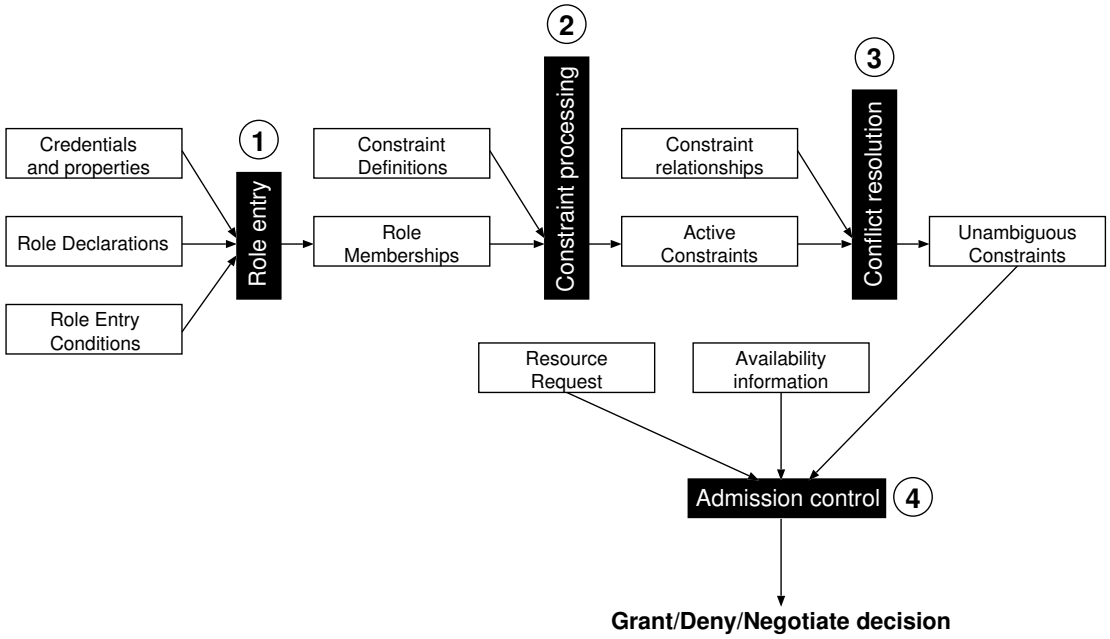

Fig. 1. Role-Based Resource Management architecture

express such policies, we use *constraint definitions*, as described in II-B.

Once the role memberships have been determined, they are associated with the constraints that apply to them. The set of constraints is reduced as constraints that are not associated with any roles are ignored further on, since they are unable to affect the admission control decision.

This process results to a set of potentially conflicting *active constraints*. It consists of the constraints that have been associated with at least one of the roles, which the user is a member of.

C. Conflict resolution

The outcome of the first two stages can result in a set of constraints that can possibly overlap or conflict when two or more constraints apply to the same resource. Then, there is a need for a mechanism to resolve such conflicts in a flexible and effective way.

To understand the difficulty of resolving conflicting resource allocation policies, one can consider the following scenarios. The resource owner specifies a policy suggesting that each customer paying by VISA card should be guaranteed 5% of the network bandwidth. The owner defines two more policies: the first one specifies that domestic customers should be guaranteed 8% of the network bandwidth as a means of attracting local users. The second one limits the use of the network bandwidth to 2% for users that do not pay their bills regularly.

For user Alice, who pays by VISA card and does not pay her bills regularly, the system would work fine if it made sure that, when constraints are conflicting, the most restrictive constraint replaces the rest. Alice would get 2% of the network bandwidth. However, for user Bob, who pays by VISA and is a domestic customer, applying the most restrictive constraint is not helpful; Bob would get 5%, while he should be getting at least 8% as any other domestic customer. It is clear that the question is not one of a trivial, technical nature, and that no global or hard-coded conflict resolution policies can be enforced.

A simple solution could be a first-match one; when a number of constraints apply to a resource, the one that was defined first overrides all others. However, this would be too inflexible, and the decision would rely on the order in which constraint definitions would be declared. Another easy fix could be to prioritize constraint definitions in a static manner, for instance by including a priority number along with each constraint definition. The problem with this approach is that it puts a very large administrative burden on the server owner, as resolving conflicts manually can become extremely difficult for large numbers of constraints.

The approach that we take is to allow the administrator to define how conflicts should be resolved explicitly, by declaring *constraint relationships*, as described in II-B. The set of active constraints is checked against the constraint relationships, and sets of conflicting constraints are replaced by single constraints and resolved.

When this stage is finished, a set of *unambiguous constraints* is produced.

D. Admission control

The resource allocation request is checked along with the current resource availability as well as the set of unambiguous constraints, in order to reach a grant/deny decision.

In order to specify the general default behaviour of the system, a simple parameter can be set to specify either that access to resources can be allowed if not explicitly

prohibited by a constraint, or prohibited if not explicitly permitted by a constraint.

Then if the request does not exceed availability or any of the constraints in the unambiguous constraints set, and if the default policy is to allow access unless a constraint prohibits it, the request is granted. If the policy is to prohibit access unless a constraint permits it, then there will have to be an explicit reservation constraint for the resource to be granted.

IV. Related work

Access control is an area that has received extensive research focus over the last three decades [10]. One of the first works establishing role-based access control as we know it today was [4], outlining the ideas of RBAC and providing a concrete formal description of role definition and membership, as well as recognizing the importance of the separation of duty problem — a refinement of those ideas, as well as an implementation, are provided in [5]. In [14], a family of well-defined RBAC models is introduced.

Several role-based systems have been devised over the last ten years, providing frameworks for the administration of roles and access rights [12], and recognising the need for defining meta-policies to resolve conflicts [11], [3]. Other researchers understand the need to combine roles and policies applied by a diverse set of sources, escaping from the "central authority" model and understanding the challenges imposed by applying RBAC to open, large-scale systems [9], [7]. A generalized version of RBAC is proposed in [2], going beyond the common subject-centric approach to role management, by allowing object-centric or environment-centric policies to be defined.

Our work uses some of the techniques invented in role-based access control, but is fundamentally different in that we use roles in a different, soft, quantitative setting, to express and apply complex resource allocation policies. In the resource management setting, roles and policies need to be *combined* instead of overriden in accordance to the "least privilege" rule, which is the solution that most RBAC systems take.

V. Conclusions

In this paper, we have proposed an architecture that uses roles to allow expressing and applying complex resource allocation policies in diverse and heterogeneous global-scale federated systems. In particular, we have seen how roles and role membership conditions can be defined, and how these can be associated with constraints in resource allocation. Also, we have devised constraint relationships as a means to resolve constraint conflicts.

References

[1] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An overlay testbed for broad-coverage services, Jan. 2003. PlanetLab Design Note PDN-03-009.

[2] M. Covington, M. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications, 2000.

[3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995, 2001.

[4] D. Ferraiolo and R. Kuhn. Role-based access controls. In *Proc. 15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[5] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *j-TISSEC*, 2(1):34–64, Feb. 1999.

[6] S. Hand, T. Harris, E. Kotsovinos, and I. Pratt. Controlling the XenoServer Open Platform, November 2002. Under submission to OpenArch'03.

[7] R. J. Hayton, J. M. Bacon, and K. Moody. Access control in an open distributed environment. pages 3–14, 1998.

[8] W. E. Johnston, D. Gannon, B. Nitzberg, L. A. Tanner, B. Thigpen, and A. Woo. Computing and data grids for science and engineering. In *Proceedings of the 2000 conference on Supercomputing*, page 52. IEEE Computer Society Press, 2000.

[9] D. Jonscher and K. R. Dittrich. Argos – A configurable access control system for interoperable environments. In T. C. Ting and D. Spooner, editors, *Database Security, IX: Status and Prospects*, pages 43–60. Chapman & Hall, 1996.

[10] B. Lampson. Protection. In *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, pages 437–443, Princeton University, 1971.

[11] E. C. Lupu, D. A. Marriott, M. S. Sloman, and N. Yialelis. A policy based role framework for access control. In *Proceedings of the first ACM Workshop on Role-based access control*, page 11. ACM Press, 1996.

[12] M. Nyanchama and S. Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgenstern, and C. Landwehr, editors, *Proc. 8th IFIP WG 11.3 Working Conference on Database Security (Database Security VIII: Status and Prospects) (Bad Salzdetfurth, Germany, Aug. 23–26*, volume A-60 of *IFIP Transactions*, Amsterdam, The Netherlands, 1995. North-Holland (Elsevier).

[13] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, Sept. 1975.

[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.