A DESIGN FOR PARAMETERIZED ROLES

Mei Ge and Sylvia L. Osborn *

Abstract Role-based access control eases the management of access control in cases where there are large numbers of objects and users. Roles provide certain access to specific data objects. In order to handle a very large number of users who each need the same access to slightly different data, we propose parameterized roles as an alternative to private roles. We motivate the requirement for parameterized roles, show how to define them, and show how with only very slight modifications, our role graph model can incorporate parameterized roles.

1. INTRODUCTION

Role-based access control has been studied for many years now [9, 4]. The role graph model is one manifestation of role-based access control [4, 5]. In some recent work [11], we looked at accessing large collections of XML documents using role graphs. One outcome of this recent work is to realize the need in some cases for parameterized roles. Such roles become necessary when a very large number of users needs access to similar data, but not exactly the same data. An example (to be expanded in Section 3) is to consider student data for a university. There may be thousands of students, and therefore thousands of objects holding student information in a database. Suppose we want to restrict each student to only seeing data concerning themselves. In our model and other RBAC models, the permissions are in terms of individual objects and access modes. In such a situation, we would have to construct individual roles for each student. The proposal in this paper is to instead extend the role graph model to include what we will call parameterized roles.

Section 2 will give some background on the role graph model. We use this version of RBAC because it has well developed algorithms which must be considered with the additions to the model. Section 3 will motivate parameterized roles with an example. Section 4 provides details of how parameterized privileges and parameterized roles will be incorporated into the role graph model.

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

A comprehensive example is given in Section 5. Conclusions are found in Section 6.

2. THE ROLE GRAPH MODEL

2.1 Basic Properties of the Role Graph Model

The Role Graph Model [3, 5] is a general access control model which allows for easier management of the assignment of permissions to users when there are very large numbers of both. It groups permissions (which we call privileges) into roles, so that by assigning a user to a role, one can assign an arbitrary number of privileges at once. It also encompasses a Group Graph Model [6] which allows users to be put into groups, which are simply sets of users. Thus, assigning a group to a role allows one to assign an arbitrary number of users to an arbitrarily large set of privileges with one operation. The Role Graph Model has similar capabilities to the RBAC models of Sandhu and others [8–1].

The Role Graph Model [3, 5] considers access control on three planes: the central plane consists of the role graph which represents role-role relationships as a role graph; the other two planes describe groups and privileges respectively. The nodes in the role graph represent roles; the edges represent the is-junior relationship between two roles. We use the term *effective privileges* to denote all privileges available through a role whether directly assigned to the role or inherited from junior roles. The *direct privileges* of role r_i are those privileges directly assigned to the role, and not available through any roles junior to r_i. A role, in turn, is a pair consisting of a role name and a set of (effective) privileges, denoted (rname, rpset). A role r_i *is-junior* to role r_j if the effective privilege set of r_i is a proper subset of the effective privilege set of r_j.

Each privilege is represented by an (object, access mode) pair, or simply (x, m). The exact nature of the object and access mode depends on the environment; for example in a relational database, the objects would be the relations and the access modes would be the allowable operations: insert, update, etc. In complex environments, there can be implications among privileges; for example the privilege to read a whole relation implies the privilege to read the individual tuples. We will have more to say about these implications shortly.

Roles are arranged in a role graph, with two distinguished roles: MaxRole and MinRole. MaxRole represents all the privileges in the role graph and need not be assigned to any user or group. MinRole represents the least privileges assigned to anyone in the system.

Role graphs have the following **Role Graph Properties**:

- there is a single MaxRole,

- there is a single MinRole,
- role graphs are acyclic,
- there is a path from MinRole to every role r_i,
- there is a path from every role r_i to MaxRole,
- for any two roles r_i and r_j, if $r_i.rpset \subset r_j.rpset$, then there must be a path from r_i to r_j.

By convention we draw the graphs with MaxRole at the top, MinRole at the bottom, and junior roles lower on the page than their seniors. We also remove transitive edges to make the graph less cluttered.

The (administrative) operations available in the role graph model are outlined in [5]. They include adding/deleting a role, adding/deleting an edge, and adding/deleting a privilege to/from a role. All of these operations abort if a cycle in the graph would be created, and restore the role graph properties given above. They all run in time polynomial in the size of the graph and the privilege sets. At the end of each of the algorithms for the role graph administrative operations, it is necessary to restore the role graph properties. This in turn involves first propagating any new privileges to senior roles, whether they have resulted from inserting a role, adding a privilege or adding an edge. Then we check for cycles by comparing effective privilege sets. We must add an edge $\mathbf{r}_i \rightarrow \mathbf{r}_j$ whenever $\mathbf{r}_i$**.rpset** $\subset$ $\mathbf{r}_j$**.rpset** (whenever $\mathbf{r}_i$**'s** effective privilege set is a proper subset of $\mathbf{r}_j$**'s** effective privilege set). Then redundant edges are removed. Note that both testing for cycles, and determining where edges are implied involve comparison operations on sets of privileges.

An example role graph is shown in Figure 1. This will be the role graph for a running example dealing with students at a university, which we use throughout the paper.

The Group Graph model [6] allows one to create sets of users, say to represent committees or people assigned to a project, who may not have the same job title. To simplify the model, each individual user is regarded as a group of cardinality 1. For example, there might be one individual, say Alice, who is the Departmental Counselor, who is assigned to the role Departmental Counselor. There might also be an admissions committee consisting of Alice, other faculty members and one student. These users can be put into a group in the group graph, and the group can be assigned to the Admissions role. The individuals in the group would also have individual and possibly other group-related role assignments.

The edges in the group graph are determined by the subset relationship between two groups. User-role or group-role assignment takes place when a user or group is assigned to a role. In [6], we discussed how the modeling of users

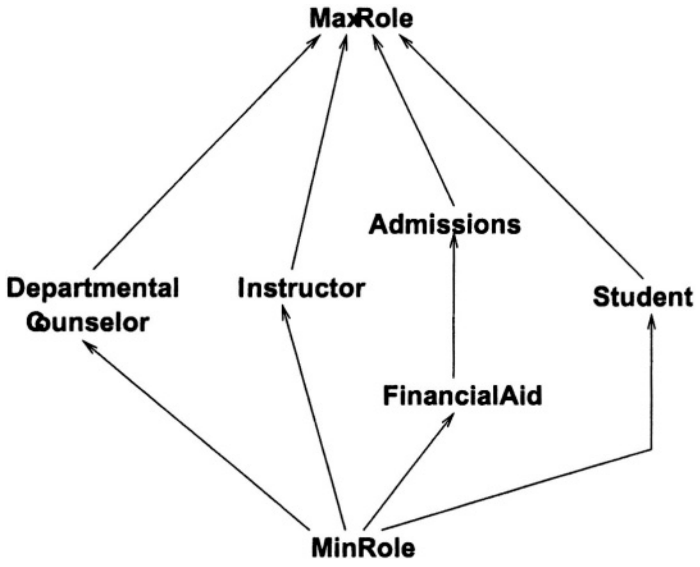

Figure 1. Example Role Graph

can be done with an object-oriented approach so that users can have attributes which might be used in user-role assignment.

2.2 Sessions

The notion of sessions was first introduced by Sandhu et al.[9]. In a session, a user is allowed to activate any subset of the roles to which they are assigned. We need to add the notion of sessions to the role graph model in order to capture what is needed to solve the problem addressed in this paper.

2.3 Privilege Propagation

In some of our previous work [2, 11], we have incorporated the methodology first suggested for object-oriented databases [7], of propagating privileges. Privileges can be represented by a triple, (s, x, m), where s is the subject, and x is the object, m is the access mode as above. For each of s, x and m, propagation can exist. As far as s is concerned, the propagation becomes the inheritance of privileges by senior roles from their juniors in the role graph. If the objects being considered are very complex, like deeply nested objects in an Object-Oriented Database, or XML documents, the propagation results from the object structure. For example, a privilege to read a whole object would propagate to the privilege to read all of its parts as well. Such propagation is not always appropriate, in which case we “turn it off” by specifying constraints which stop the propagation. These constraints play the same role as negative permissions in [7], where negative permissions have precedence over positive ones.

As well as the propagation due to object structure, it is possible to have propagation due to the m part of (s, x, m). In some systems, for example, having a write privilege implies the same subject also has a read privilege on the same object.

In what follows, then, we assume that the role graph model has been enhanced with privilege propagation, which can be specified by giving a schema which guides the object-based propagation, and another graph which captures the access mode-based propagation. In order for the role graph algorithms to produce a correct role graph, we need to assume that whenever a new privilege is added to a role, first all implications of this privilege are calculated, and then this total set of privileges is added to the role. The function of restoring role graph properties then propagates these privileges to any senior roles, unless prevented by constraints.

3. MOTIVATION FOR PARAMETERIZED ROLES

To motivate our notion of parameterized roles, consider the following example. We have data concerning students, which is in a database. It happens that we have in mind a (large) collection of XML documents concerning students at a university, whose structure is shown in Figure 2. The node labelings in this figure are the XML tags which would be defined in an XML schema or DTD. Figure 3 shows some possible instances of this data. In Figure 2, the data in dashed ovals can appear an arbitrary number of times within its parent element, and the data in solid ovals can appear at most once. Rectangles denote XML attributes which also can appear at most once. In Figure 3, XML attribute values are shown in quotes, and element values are shown unquoted.

Consider again the role graph for typical users within a university community as given in Figure 1.

Our intention is that students should be able to access information concerning themselves, that instructors can access course related information for courses that they teach, and that users assigned to the Financial Aid role can see all FinancialInfo as well as the students' names. The Financial Aid role should be able to see all students' financial information, whereas a student should only be able to see their own information. The privileges in an RBAC system are of the form (x, m) where x refers to a particular object, and m to one of its valid access modes. For the Financial Aid role, the object reference could be to whatever container holds the financial information for all students. For a Student role, however, the object part of the privilege has to isolate the data for one student.

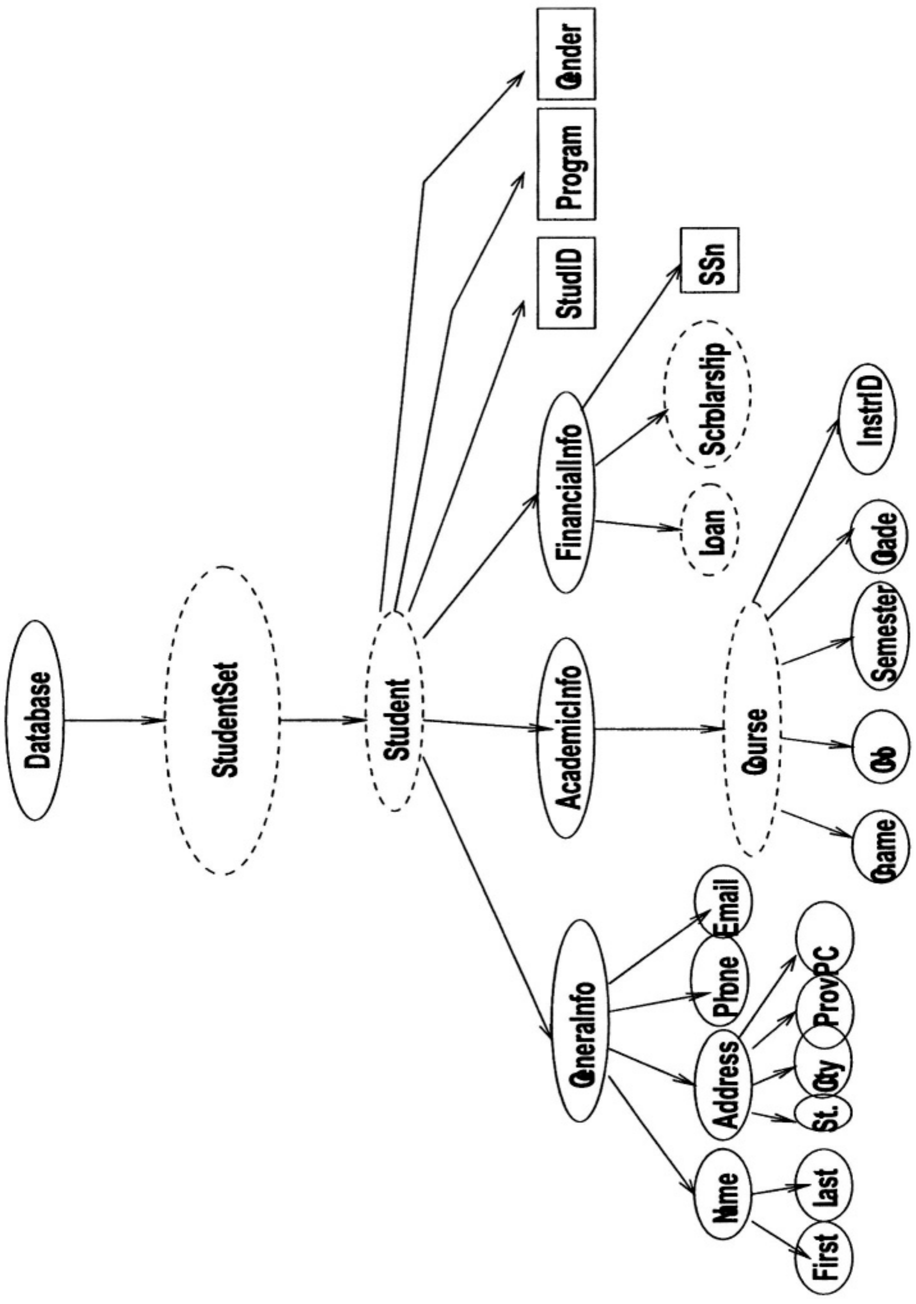

Figure 2. Data Schema for our Example

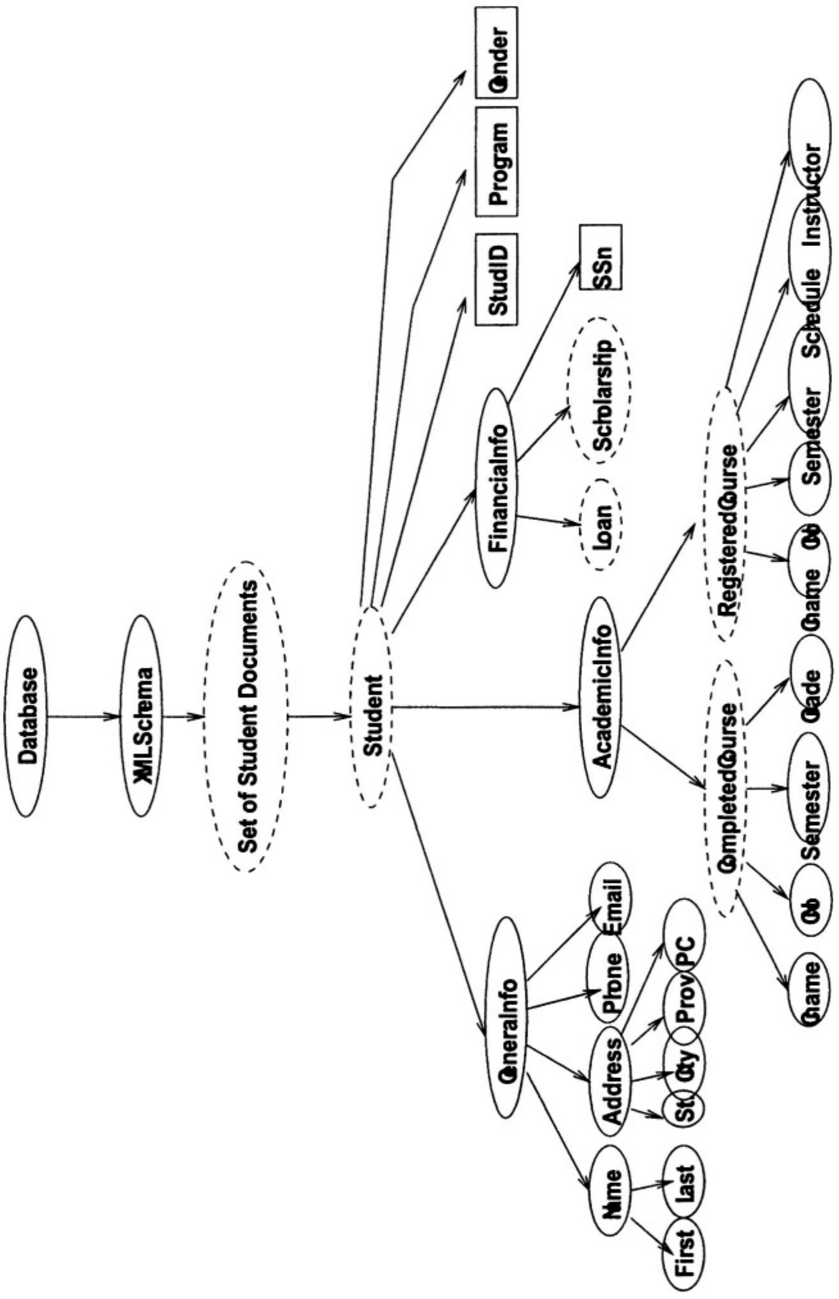

Figure 3. Data Instances for our Example

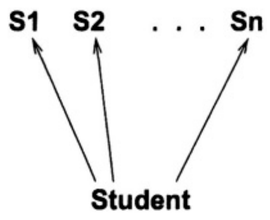

Figure 4. Individual Private Roles for Students

One suggestion often made in a situation like this is to have private roles, one for each student, such that each is senior to the original Student role, as shown in Figure 4. Privileges common to all students would be assigned to the general Student role, to be inherited by the individual S1, S2, etc. roles, and privileges accessing private information for individual students would be assigned to the individual roles. Clearly with a large number of students, this solution would not be workable – some security administrator would have to build each of the private roles.

A similar case can be made for departmental counselors. They should be able to see the academic records of all students in their department (e.g. all Computer Science students), which may be a large number but still not all the student records in the database. The number of private roles needed here is on a smaller scale but nevertheless may be required by university policy.

We intend to express the privileges in our examples using an XPath-like syntax [10]. It should however be noted that XML and XPath are not essential to the basic idea of parameterized roles. If the above data were stored in a relational database, for example, there would be a relation for FinancialInfo, to which the FinancialAid role would be given complete read access. The student information would be stored in one or more relations. We would still need a query to isolate those tuples relating to a specific student and make visible to each student their own data and no other student's data.

Note that in the above examples, the differentiation in what each student gets to see lies in the object part of a privilege, not in the access mode part. If the role allows the student to read and update their personal information, these same access modes are intended to be made available to all students through their private roles. Only the data touched varies.

4. PARAMETERIZED PRIVILEGES AND ROLES

4.1 Parameterized Privileges

We begin by discussing parameterized privileges. An ordinary privilege in the role graph model consists of a pair, (x, m), where x is a specific (single) object and m is an access mode. Its presence in a role means any users assigned to this role can perform this operation when the role is active in a session. Such a privilege might be expressed, using XPath, as (//Student2/GeneralInfo, read), i.e. read the GeneralInfo for a particular student, Student2, whose name is Paul Jones. According to the propagation of privileges [2], if read propagates down in the object graph (Figure 3 here), then the permission to read the GeneralInfo of Student2 also implies that the read on all child nodes will also be added to the privilege set of any role to which this is assigned. This propagation can be turned off by a constraint, for example if the privilege to read email addresses is forbidden to the role which is under consideration, then the read privilege for the email node is not added to the role.

A parameterized privilege will have the form: $(x \mid x\{a_1, a_2, \ldots, a_n\}, m)$, where the object is either simple, denoted by x, or isolated by one or more parameters, denoted by $x\{a_1, a_2, \ldots, a_n\}$. Each parameter, in turn, is denoted by a pair, (a_{name}, a_{domain}) where a_{name} is the name and a_{domain} is the domain of the parameter. Each parameter is a place to hold a variable that has changing values from a predefined domain.

These parameter values will be supplied when a user activates a role in a session. It is possible to view users as objects with attributes which they present when activating a role. Further discussion of these details is beyond the scope of this paper.

For the remainder of this paper, we will regard a privilege such as (PersonalInfo{StudID}, update) to correspond to a privilege containing an XPath-like expression: (//PersonalInfo[StudID = value], update), where the value is supplied at run time and the XPath expression is executed to provide the private data for this activation of the role.

4.2 Parameterized Roles

Given that some or all of the privileges in a role might be parameterized, roles themselves can be regarded as parameterized. Thus, rather than referring to a role as (rname, rpset), we now use the notation (rname, rpset, rparamset) where rparamset is empty for roles which have no parameterized privileges. In this case, rparamset contains the union of all the parameters used in any parameterized privileges in rpset. Note that in a role graph, when a junior role has one or more parameterized privileges, then its senior roles must also be parameterized roles as they inherit the parameterized privileges. Of course, a role graph can contain a mixture of parameterized roles and ordinary roles.

4.3 Sessions

The importance of sessions should not be overlooked. As pointed out in [9], the user-role assignment relationship is many to many, and represents "can perform". It is the session which is associated with a single user. Thus to handle the situations we have in mind here, we need to be able to isolate a single user and supply their parameter values before activating the parameterized roles. The idea of a session provides exactly the right properties for our solution.

4.4 Role Graph Operations

As we have noted above, many of the role graph operations contain comparisons of sets of privileges. We need to extend the definition of when a privilege belongs to a privilege set, when parameterized privileges are involved.

Definition *Privilege Inclusion:* Given privilege p and privilege set S,

- if p is not parameterized and is exactly the same as p' in S, then $p \in S$.
- if p is of the form $p = (x\{a = v\}, m)$, and $\exists p' \in S$ such that $p' = (x\{a'\}, m)$ where $a'_{name} = a$ and $a'_{domain} = a_{domain}$, then $p \in S$.

Given this definition, testing if one privilege set is a subset of another follows: i.e. $S_1 \subseteq S_2$ iff $\forall p \in S_1, p \in S_2$. $S_1 \subset S_2$ and $S_1 = S_2$ are defined in the obvious way.

We do not want two parameters with the same name, with different semantics or different domain names to simultaneously exist in the same privilege or in the same role. Therefore the following tests must be added to the privilege insertion algorithm:

1 If a parameterized privilege has more than one parameter, the parameters must have distinct names.

2 If a parameterized privilege has a parameter with the same name as another parameter in the role, the two parameters must have the same domain.

These additional checks need to be added to the privilege addition algorithm. Otherwise, by defining $\in$ and $\subset$ for parameterized privileges and privilege sets, we can use the same algorithms as before for the role addition/deletion, edge addition/deletion and privilege addition/deletion algorithms which operate on role graphs.

5. EXAMPLE

To demonstrate the notion of parameterized roles, we now present a more complete example based on the role graph and XML structures presented in Figures 1, 2 and 3. In this example, we assume that given a new privilege, the propagation of this privilege down the schema graph as shown in Figure 2 is carried out by the privilege insertion algorithm [2, 11]. We have not given the graph showing the propagation due to access modes, so all necessary privileges with different access modes are explicitly given here.

A student can read his or her individual general information; update his or her individual general information except for name; read academic information, financial information and personal attributes pertaining to him or her self. For Role Student, the direct privileges are:

{p1 = (//Student[@StudID = param1]/GeneralInfo, update),
p2 = (//Student[@StudID = param1]/GeneralInfo, read),
p3 = (//Student[@StudID = param1]/AcademicInfo, read),
p4 = (//Student[@StudID = param1]/FinancialInfo, read),
p5 = (//Student[@StudID = param1]/@StudID, read),
p6 = (//Student[@StudID = param1]/@Program, read),
p7 = (//Student[@StudID = param1]/@Gender, read)}

As well, a constraint is defined for this role:

{c_{stu} = (//Student[@StudID = param1]/GeneralInfo/Name, update)}

which means that the update privilege does not propagate to the Name subelement within the Student's GeneralInfo.

The instructor role is also a parameterized role. Instructors can read certain student information and read and update the grades of the students that they teach. For Role Instructor, the direct privileges are:

{p8 = (//Course[/InstrID = param2], read),
p9 = (//Course[/InstrID = param2]/Grade, read),
pl0 = (//Course[/InstrID = param2]/Grade, update),
p11 = (//Student[//InstrID = param2]/Name, read),
p12 = (//Student[//InstrID = param2]/@StudID, read),
p13 = (//Student[//InstrID = param2]/@Program, read),
p14 = (//Student[//InstrlD = param2]/@Gender, read)}

The Departmental Counselor role can read all general information, academic information of students in their department's program, and create, update and read the course information. The parameter value in this case matches the program the student is in. For the Departmental Counselor role, then, the direct privileges are:

{p15 = (//Student[@Program=param3]/AcademicInfo/Course, create),
p16 = (//Student[@Program=param3]/AcademicInfo/Course, update),
p17 = (//Student[@Program=param3]/AcademicInfo, read),

p18 = (//Student[@Program=param3]/GeneralInfo, read),
p19 = (//Student[@Program=param3]/@StudID, read),
p20 = (//Student[@Program=param3]/@Program, read),
p21 = (//Student[@Program=param3]/@Gender, read)}

The FinancialAid role can update and read financial information of all the students in the university and read the information of all the students in the university. This is a general role without parameters. Its direct privileges are:

{p22 = (/StudentSet, read),
p23 = (/StudentSet//FinancialInfo, update)}

This role also has a constraint:

{c_{fn} =(//Student/FinancialInfo/SSN, update)}.

The Admissions role creates new students as they are admitted to the university. This is not a parameterized role. Its direct privileges are:

{p24 = (//Student, create), p25 = (/StudentSet, read), p25 = (//Student, create), p26 = (//Student, update)}

This role also has the constraint:

{c_{amd} =(//Student/FinancialInfo/SSN, update)}.

In most of the above roles, a different method can be used to achieve the same purpose. For example, in the Department Counselor role, we could keep p15 and p16, give a read privilege to the whole student element, and use a constraint to deny read to the FinancialInfo element. The current version of our role graph tool, which incorporates the work in [2, 11], labels each privilege as either given or implied, and allows the user to open a window in which to see all the privileges that result from propagation. In this way the security designer can verify that their intended security design has been entered correctly into the role graph tool.

6. CONCLUSIONS

Adding parameters to privileges and roles greatly extends the usefulness of the role-based access control models. In a situation like that found at our university, with over 25,000 students, having to implement private roles for each student with slightly different access requirements is completely impractical. Extending the role graph model with parameterized roles required including the notion of sessions, and only slight modifications to the rest of the model. With the exception of a small modification to the privilege insertion algorithm, and enhancing the test for membership of a privilege in a privilege set, the algorithms already developed for the role graph tool can be used without modification.

A good role graph design can go a long way toward achieving the principle of least privilege, where users get access to what they need and nothing extra. Parameterized roles enhance this by providing very specialized privileges to users, where the exact privileges are decided at run time when a session is started. One could say this adds active security characteristics to RBAC.

The model presented in this paper can also be considered to be a high level model for all those systems which allow clients to view their private data in web applications. The user has to log in, and presumably a database query isolates the data involving the individual client. The design of the different user interfaces is represented by the notion of parameterized roles. The privileges given to such sessions would currently be buried in application code. Our model of parameterized roles might give designers of such systems a more systematic way of viewing their problems.

At a lower level, one might worry that the privilege sets in the roles are getting too big. As we have noted above, our algorithms must be able to reason about the total effective set of privileges for the roles, after propagations. Whether or not we actually store them this way is a decision to be made at implementation time. In some sense, this is due to the size of the problem, i.e. the size of user set and the size and complexity of the database. We have provided a way to shelter the system from the size of the user set when parameterized roles are appropriate, but as far as the data is concerned, XML documents can be very complex, and if we truly want to deal with fine grained and complex access control situations, we have to be prepared to have very large privilege sets. In an application with only read operations, there still might be a complex pattern of which elements users are and are not allowed to see. We could just store the read access on the topmost node, and expand the privilege when necessary, but we feel that by the security designer being able to see what privileges have been propagated, they receive some useful feedback.

References

[1] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC,* 4(3):224–275, 2001.

[2] Cecilia M. Ionita and Sylvia L. Osborn. Privilege administration for the role graph model. In *Research Directions in Data and Applications Security, Proc. IFIP WG11.3 Working Conference on Database Security,* pages 15–25. Kluwer Academic Publishers, 2003.

[3] M. Nyanchama. *Commercial Integrity, Roles and Object Orientation.* PhD thesis, Department of Computer Science, The University of Western Ontario, London, Canada, Sept. 1994.

[4] M. Nyanchama and S. L. Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database Security, VIII, Status and Prospects WG11.3 Working Conference on Database Security,* pages 37–56. North-Holland, 1994.

[5] M. Nyanchama and S. L. Osborn. The role graph model and conflict of interest. *ACM TISSEC,* 2(1):3–33, 1999.

[6] S. Osborn and Y. Guo. Modeling users in role-based access control. In *Fifth ACM Workshop on Role-Based Access Control,* pages 31–38, Berlin, Germany, July 2000.

[7] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans Database Syst,* 16(1):88–131, 1991.

[8] R. Sandhu, V. Bhamidipati, and Q Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. on Information and Systems Security,* 2(1):105–135, Feb. 1999.

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

[10] w3c. XML path language (XPath) 2.0, W3C working draft 15. Technical report, http://www.w3.org/TR/xpath20, Nov. 2002.

[11] Jingzhu Wang and Sylvia L. Osborn. A role-based approach to access control for XML databases. In *Proc. ACM SACMAT,* 2004.