A FORMAL MODEL FOR PARAMETERIZED ROLE-BASED ACCESS CONTROL

Ali E. Abdallah and Etienne J. Khayat
Research Institute for Computing,
London South Bank University,
103 Borough Road,
London SE1 0AA, U.K.
{A.Abdallah@lsbu.ac.uk, E.Khayat@lsbu.ac.uk}

Abstract Role-Based Access Control (RBAC) usually enables a higher level view of authorization. In this model, access permissions are assigned to roles and, in turn, roles are allocated to subjects. The usefulness of the RBAC model is well documented. It includes simplicity, consistency, scalability and ease of manageability. In practice, however, only limited versions of RBAC seem to have been successfully implemented, notably in applications such as databases and operating systems. The problem stems from the fact that most applications require a finer degree of authorization than what core RBAC models are able to provide. In theory, current RBAC models can be adapted to capture fine grained authorizations by dramatically increasing the number of distinct roles in these models. However, this solution comes at an unacceptably high cost of allocating low level privileges which eliminates the major benefits gained from having a high level RBAC model.

This paper presents a methodology for refining abstract RBAC models into new Parameterized RBAC models which provide finer grain of authorizations. The semantics of the Parameterized RBAC model is given as a state-based core RBAC model expressed in the formal specification notation Z. By systematically applying this methodology the scope of applications of RBAC is substantially extended and the major benefits of having the core model are maintained.

1. Introduction

RBAC is an access control mechanism based on the rationale that access rights are assigned to roles, rather than to the subjects that perform these roles [1–5]. This approach is attractive for concisely describing authorization, particularly within organizations, because responsibilities are often assigned to employees (subjects) based on their duties (roles). RBAC is also very useful when it is

adapted to fit the organizational structure of an institution because it bridges the gap between its functional requirements and the technical authorization aspects of its security policy. Hence, offering a high level view of authorization and its operational management. However, few examples can be found where only limited versions of RBAC are applied, namely in databases and in operating systems such as Solaris [7, 8]. The reason for this is that the direct application of abstract RBAC requires significant expressive power to cater for access requirements in large organizations. For instance, the way permissions are currently described in RBAC suggest that every role has a unique set of permissions, which are assigned to its associated subjects or principals. With this, two different subjects, occupying exactly the same role, will have identical permissions. This might not be desirable because every subject exercises its permissions in the context of its own duties [9]. This type of definition makes a successful direct application of RBAC requiring large amount of work to express access rights. Its success might be more related to the advanced techniques provided by the implemented application, such as the "views" in databases [6], rather than the features of the adopted RBAC. To clearly illustrate the lack of expressiveness in a direct application of RBAC, consider the case of online banking. In this case, the clients are the *subjects* and the bank accounts are the *objects*, and the role to describe the clients who have bank accounts is referred to as *Account_Holder*. Although this role applies to all bank clients, which can reach hundreds of thousands, the access permissions associated with every client (subject) occupying this role should be different. When using the online banking service, every client can only access its own account details, and not those of other clients occupying the role *Account_Holder*. It is possible to express the permissions for such clients by directly applying RBAC, but only at a considerable cost:

Firstly, although roles such as *Account_Holder* are defined as a single role, their implementation suggests their instantiation into a large number of roles to cater for every client, which presents a huge burden on the intellectual manageability of access rights.

Secondly, the direct implementation of RBAC reduces the scalability of the mechanism in large organisations because instances of role have to be treated as different roles instead of being grouped under a single definition.

Thirdly, the consistency of the distribution of access rights is affected because similar roles will have to be treated differently and managed separately.

These advantages, generally associated with RBAC models, can be maintained if this case study is modelled as a parameterized RBAC. In this model, core RBAC components, such as roles, would depend on values of a parameter. To extend RBAC into a parameterized model, data about the values of the parameters should be provided. New permissions that might be created due to the parameterization should also be identified. Hence, the construction of the

Parameterised RBAC (*PRBAC*) results from the combination of the new components as follows:

$$RBAC + Parameters\ Data + New\ Parameterized\ Permissions \Longrightarrow PRBAC \quad (1)$$

When the direct implementation of the core RBAC results in many drawbacks, the Parameterised RBAC achieves the same effect and avoids these drawbacks. Reconsidering the online banking example, the role *Account_Holder* can be generalised to a parameterized role $(Account\_Holder, m)$, where m is an account number. This role can later be instantiated to each account to give the appropriate instances of the permissions. Following this, there will be a single general parameterized definition for a role. This parameterization can be repeated to support further levels of granularity by including additional parameters such as bank branch. In this case, a role would be defined as $((Account\_Holder, m), m_1)$, where m_1 is the bank branch number. This series of "nested" parameters can be related in a hierarchy that depends on the type of the used parameters and the way the parameterization is performed. This hierarchy is not to be confused with the role hierarchy such as job positions in organisation.

In this paper, we present a rigorous formal model for a parameterized RBAC [4], in the Z notation [12–14]. We refer to it as *PFRBAC* because it is derived from an extension of *FRBAC*, a flat RBAC presented in a previous work by the authors of this paper [5]. Being formal, *PFRBAC* presents a clear and mathematically concise way for the implementation of *FRBAC*. It is detailed in its modelling of the components and features of a Parameterised RBAC and complete in its presentation of the necessary semantics. With its type of a general parameterized definition of RBAC components, this model enables a good and comprehensive intellectual manageability of access rights and provides a consistency in their distribution. The support of a hierarchy of parameters helps to achieve an extremely fine granularity in access control, which is difficult to achieve in a non-parameterized RBAC.

The remainder of this paper is structured as follows. Section 2 provides a brief overview of related work on Parameterised RBAC. The parameterized model is constructed in Section 3. This section details the derivation of the parameterized concepts from the core ones. It presents the formal state based description of the model in Z and shows the process of building a nested parameterized model from an hierarchy of parameters. Section 4 presents a discussion of the effect of the choice of the parameterized concepts on the expressiveness of the model and Section 5 concludes the paper.

2. Related Work

There has been few attempts to model the Parameterised RBAC in the contexts of many works [9, 11, 15]. One of these attempts included a very useful

definition of the parameterized role [11]. This definition was general and very expressive, however, the rest of this work did not reflect its usefulness because it was restricted in terms of the presented semantics and did not expand in a way that enables a useful way of implementing the Parameterised RBAC. This attempt did not also present full formal semantics of a parameterized RBAC which treat all its features and concepts, such as the parameterized roles, permissions and objects. The relation between the RBAC concepts in this attempt and the ones in the widely known RBAC models such as [1–5] is not explicit.
Another parameterization of the definition of *role* was presented in [15]. This work aimed to suit the problem of controlling access to the contents of objects in RBAC and focused on databases as its area of application. Although this might be useful in the mentioned applications (databases), it drifts from the concepts of RBAC models such as the RBAC standard [4], *RBAC*96 [1] and *FRBAC* [5] which consider the object as the primitive unit that can be assigned access rights and do not model its content. The focus of the parameterized RBAC to solve a particular problem has undermined the generality of the approach because of its limited applicability to cases outside the presented scope.
Another definition of *role* as *a series of policy statement and constraints* has been proposed when constructing another solution-specific version of a parameterized RBAC [9]. The data type definition for roles has been substituted for role classes in order to be instantiated. Because of these changes in the concepts, it was difficult see how the commonly known RBAC models, such as the ones in [1, 4, 5], would fit in this work.
All these attempts did not present a methodology to guide into the parameterization of RBAC. The parameterized RBAC concepts have been defined without specifically mentioning their original RBAC definitions. This makes it difficult for implementers to derive a Parameterised RBAC from an existing RBAC model because the parameterization would need a significant amount of work. It also undermines the advantages of scalability, consistency and ease of manageability that a Parameterised RBAC offers.

3. Model for the Parameterized RBAC

The parameterized RBAC model presented in this paper is based on an extension of *FRBAC* [5]. It supports the commonly known RBAC adopted concepts and definitions [1–5], and the results of this paper equally apply to the core model in the RBAC standard [4]. This section presents the methodology to construct a parameterized RBAC from *FRBAC* and a supply of data about parameters and of newly created permissions (according to (1)). One important property of this approach is that the parameterization can be done several times and successively in order to obtain very fine grained access control. It

turns out that this property of nesting is directly associated with the hierarchy of data representing the parameters.

3.1 The Flat RBAC

The components of a flat RBAC model are derived from the following entities: ROLE, OBJECT, SUBJECT, PRINCIPAL, OPERATION, TASK, PERMISSION, which are respectively the sets (or data types) for all roles, objects, subjects, principals, operations, tasks and permissions. The sets of components of an implemented RBAC model are referred to as: *Roles*, *Objects*, *Subjects*, *Principals*, *Operations*, *Tasks* and *Permissions*. For instance, *Roles* is the set of roles that are defined in the organisation implementing RBAC. The same analogy applies to the other components. *FRBAC* is summarised in the following schema [5].

Referring back to the online banking example. *Subjects* would be the clients and the employees of the bank, and *Principals* would the set of their associated usernames. We consider four roles in this example:
Account_Holder, *Manager*, *Clerk*, *System_Administrator*. To illustrate the initialisation of an RBAC model for the case of the online banking application, we use this following toy example. We assume that the sets of all accounts and pin numbers are referred to respectively as *Accounts*, and *Pins*. The " $'$ " sign is a convention for the initialisation of the components in the **Z** notation [14].

FRBAC

$$
\begin{array}{l}
Roles : \mathbb{P}\, ROLE \\
Principals : \mathbb{P}\, PRINCIPAL \\
RoleAllocation : PRINCIPAL \rightarrow \mathbb{P}\, ROLE \\
Subjects : \mathbb{P}\, SUBJECT \\
SubjectAssociation : SUBJECT \rightarrow \mathbb{P}\, PRINCIPAL \\
SubjectRole : SUBJECT \rightarrow \mathbb{P}\, ROLE \\
Objects : \mathbb{P}\, OBJECT \\
Operations : \mathbb{P}\, OPERATION \\
Tasks : \mathbb{P}(OPERATION \times \mathbb{P}\, OBJECT) \\
Permissions : ROLE \rightarrow \mathbb{P}\, TASK \\
\hline
\mathrm{dom}\, RoleAllocation = Principals \\
\mathrm{ran}\, RoleAllocation \subseteq Roles \\
\mathrm{dom}\, SubjectAssociation = Subjects \\
\mathrm{ran}\, SubjectAssociation \subseteq Principals \\
\mathrm{dom}\, SubjectRole = Subjects \\
\mathrm{ran}\, SubjectRole \subseteq Roles \\
\mathrm{dom}\, Permissions = Roles \\
\bigcup \mathrm{ran}\, Permissions \subseteq Tasks
\end{array}
$$

Initialisation of an Example State:

Roles′= {Account_Holder, Manager, Clerk, System_Administrator}
Principals′= {c_1, c_2, c_3, c_4, john_1, ema_1, ema_2, denise_1}
RoleAllocation′= {(c_1, {Account_Holder}), (c_2, {Account_Holder}), (c_3, {Account_Holder}), (c_4, {Account_Holder}), (john_1, {Clerk}), (ema_1, {Manager}), (ema_2, {Manager, Clerk}), (denise_1, {System_Administrator}) }
Subjects′= {Anne Roling, Mike Lowe, John Brown, Ema Thomas, Denise Logan}
SubjectAssociation′= {(Anne Roling, {c_1}),(Mike Lowe, {c_2}),(John Brown,{john_1, c_3}), (Ema Thomas, {ema_1, ema_2}), (Denise Logan, {denise_1, c_4})}
SubjectRole′= {(Anne Roling, {Account_Holder}), (John Brown, {Clerk, Account_Holder}), (Ema Thomas, {Manager, Clerk}),(Mike Lowe, {Account_Holder}), (Denise Logan, {System_Administrator, Account_Holder})}
Objects′= { Accountnumbers: P(N), Accounts: N → Account, Pins: N → Pin }
Operations′= { Create, Deposit, Withdraw, View, Transfer, Assign, Backup}
Tasks′= { (Create(n:N, a: Account, p:Pin), {Accountnumbers, Accounts, Pins}), (Deposit(k:N, n:N),{Accounts}, (Withdraw(k:N, n:N), {Accounts}), (View(n:N),{Accounts}), (Transfer(k:N, n_1, n_2:N), {Accounts}), (Assign(n:N), {Pins}), (Backup, {Accountnumbers, Accounts, Pins}) }
Permissions′= {(Manager, {Create, Deposit, Withdraw, View, Transfer, Assign}), (Clerk, {View, Deposit, Withdraw}), (Account_Holder, {}), (System_Administrator, {Backup}) }

3.2 Construction of the Parameterized Model

Parameterizations. By associating privileges with roles instead of principals, RBAC offers a scalable means for expressing access control. The size of the privilege table grows proportionally to the number of roles (which is usually small) as opposed to the number of principals (which can be very large indeed). However, in practice, most real applications require a finer grain of access control. As we have seen in the previous example of a pure RBAC model of a Bank , we were able to fully specify access control for some roles such as Manager and Clerk but not for other roles such as Account_Holder. The privileges of two different customers holding the Account_Holder role are not identical. Hence, new information needs to be added to the model to correctly capture the appropriate privileges for this role. To overcome this limitation, we propose parameterizations of the RBAC model. The effect of parameterizations is usually to enlarge the size of one of the components of the pure RBAC model, that is *roles*, *objects*, or *tasks*. This aim is achieved by the addition of parameters to values in one of the RBAC components or their attributes, i.e. *roles*. Hence, in the refined parameterized RBAC model of a Bank, the Account_Holder role will no longer exist! It will be replaced by several instances of Account_Holder(m) where m is a parameter (variable) drawn from an appropriate set of values.

Choice of Parameters. The choice of parameter and the set of values from which it can be instantiated is application dependent. The overriding objective is to be able to adequately and fully capture the privileges of each role in the refined model. In theory, the values from which a parameter can be instantiated is just a set of abstract labels and does't have to have any meaning! The purpose is to refine a single entity (such as a role name) into a set of labelled (or "colored") alternatives. In practice, however, the parameter may correspond to the focus of access control (object or subject), may reflect an underlying concept such as ownership (files and accounts) or the primary key in a relational database, or may reflect a level in the organizational hierarchical structure for managing the application (faculty, department, course or module). Let *PARAMETER* be the type of the required parameter, say a variable m, and let M be the set values from which it can be drawn in the application. We have:

$$m : PARAMETER;\ M : \mathbb{P}(PARAMETER);\ m \in M \tag{2}$$

In the banking example, since objects are parameterized by the account numbers currently allocated to clients, the set *accountnumbers*, it will be useful to use the same values to parameterize the Account_Holder role. Hence, the new role Account_Holder(m) denotes the role of holding a specific account, namely that whose number is m.

$$M = \{n\_1, n\_2, n\_3, n\_4\} \tag{3}$$

Generating refined Roles for Parameterised RBAC. Having chosen an adequate parameter and identified the range of its possible values, the next step is to identify what RBAC components, such as objects and roles, to parameterize. The most useful candidate for parameterizations is usually the *Roles* component. The objective is to fully define the privileges for each role in the model. Those roles for which access control is fully defined in the core RBAC model, however, parameterizations may not be appropriate and will not bring any benefits. Therefore, not all the values in a core RBAC component can be parameterized. Hence, our approach is to split the content of each core RBAC component into two parts: those which will be replaced by the refined parameterized versions (*ParamComponent*) and those which will be left unchanged (*Component* − *ParamComponent*). The process of generating the *Roles* component in the Parameterised RBAC model is illustrated in Figure 1. The set of roles, *PRoles*, in the new model is derived as:

$$PRoles = (Roles - ParamRoles) \cup (ParamRoles \times M) \tag{4}$$

The type for a role in the parameterized RBAC model can be inferred as:

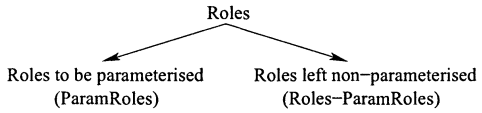

Figure 1. Division of *Roles* for parameterization.

$$PROLE = ROLE \uplus (ROLE \times PARAMETER) \tag{5}$$

For ease of readability, we adopt the following syntactic convention:

$$(c, m) \equiv c(m) \tag{6}$$

where m is a parameter and c is an RBAC component such as a role or an object. In the Banking application, for instance, only the role of *Account_Holder* is to be parameterized; hence, $ParamRoles = \{Account\_Holder\}$. All the other roles, $(Roles - ParamRoles)$, will migrate unchanged into the parameterized model. The set *PRoles* can be calculated as follows:

$$\begin{aligned} PRoles = &\{Account\_Holder(m) \mid m \in \{n\_1, n\_2, n\_3, n\_4\}) \\ &\cup \{Manager, Clerk, System\_Administrator\} \end{aligned} \tag{7}$$

Please note that *Account_Holder* is no longer a role in the new model.

Generating Subjects and Principals for Parameterised RBAC. Subjects refer in general to human users [4, 5] and may not be useful for parameterizations. Because parameterizations only addresses the issue of fully capturing access control, it would seem odd to allow the underlying set of human users to change from the core model. Therefore, we have taken the view that the *Subjects* component should remain unchanged after parameterizations.

The same reasoning may not necessarily apply to the generation of the set *Principles*, that consists of usernames and public keys acting on behalf of users. Parameterizations of principles may lead to a classification of usernames upon which aspects of access control could be determined. Therefore, by analogy with the role parameterizations, (Figure 1), the *PPrincipals* component of the parameterized model can be calculated as follows.

$$PPrincipals = (Principals - ParamPrincipals) \cup (ParamPrincipals \times M) \tag{8}$$

The data type of principals in the Parameterised RBAC can be inferred as:

$$PPRINCIPAL = PRINCIPAL \uplus (PRINCIPAL \times PARAMETER) \tag{9}$$

In the banking example, the set of principles remains the same as in the core model. That is, $ParamPrincipals = \{\}$.

$$PPrincipals = \{c\_1, c\_2, c\_3, c\_4, john\_1, ema\_1, ema\_2, denise\_1\} \tag{10}$$

Generating refined Objects for the Parameterised RBAC. the *PObjects* component of the parameterized model can be calculated as follows.

$$PObjects = (Objects - ParamObjects) \cup (ParamObjects \times M) \tag{11}$$

The data type of objects in the Parameterised RBAC can be inferred as:

$$POBJECT = OBJECT \uplus (OBJECT \times PARAMETER) \tag{12}$$

In the banking example, the set of objects remains unchanged. That is, $ParamObjects = \{\}$.

$$PObjects = Objects \tag{13}$$

Generating Permissions for Parameterised RBAC. First the tasks component in the parameterized model should take into consideration the changes in the object component. If the object component remains unchanged then the tasks component will also remain unchanged. Hence, in the banking example, we have:

$$\begin{gathered} PTasks = \{(Assign, p : Pins), (Create, a : Accounts), (Deposit, a : Accounts), \\ (Withdraw, a : Account), (Transfer, [a_1 : Account, a_2 : Accounts]), \\ (View, a : Accounts)\} \end{gathered} \tag{14}$$

What would only change in this case is the association of these tasks to new roles (the parameterized ones), which are the permissions. A parameterized permission relates a parameterized role to its authorized tasks. In a Parameterised RBAC, the permissions include 3 types:
Firstly, *ParamPermissions*, the permissions of RBAC parameterized as a result of the instantiation of the roles that *would be parameterized*, associated with the authorized tasks, and defined as:

$$\begin{gathered} ParamPermissions = \{(r(m), p) \bullet \\ (r, p) \in Permissions \wedge r \in ParamRoles \wedge m \in M\} \end{gathered} \tag{15}$$

In the online banking example, these permissions are:

$$\begin{gathered} ParamPermissions = \{(Account\_Holder(m), (View, Accounts(m))), \\ (Account\_Holder(m), (Withdraw, Accounts(m))), \\ (Account\_Holder(m), (Transfer, \{Account(m), a_2 : Account\}))\} \end{gathered} \tag{16}$$

Secondly, the permissions of the RBAC model that need not be parameterized, i.e. the permissions of RBAC whose domain of application is restricted to

Roles−ParamRoles. These permissions are defined as: $(Roles-ParamRoles?) \lhd Permissions$. In the online banking case, they are listed as:

$$PPermissions = Permission \cup ParamPermissions \tag{17}$$

Thirdly, new permissions that result due to the required private accesses for some parameterized components such as roles. Again, the data type of parameterized parameters can be deduced as:

$$PPERMISSION = PROLE \times TASK \tag{18}$$

Generating the Parameterised RBAC Model:. As shown in the previous example, a non-parameterized RBAC cannot capture permissions such as the one that authorise a bank client to view his own account $(Account\_Holder, (View, a : Accounts))$, because it cannot guarantee that a is exactly the account of the client requesting to view it. In this case, a parameterization of RBAC is needed. To be accomplished, this procedure requires the following entities to be provided:

1 The non-parameterized RBAC model, containing all the declarations and concepts of RBAC shown in $\langle 1 \rangle$ in the schema called *Parameterize_RBAC*. Note the **Z** convention to postfix the symbol "?" after a variable's name to denote an input, and to postfix the symbol "!" after a variable's name to denote an output.

2 The list of parameters M, shown in $\langle 2 \rangle$.

3 The list of components to be parameterized, namely *ParamProles*, *ParamPrincipals* and *ParamObjects* respectively defined in $\langle 3 \rangle$, $\langle 4 \rangle$ and $\langle 5 \rangle$. These are derived from the components of the RBAC model that need to be parameterized.

4 The newly induced private permissions, referred to as *New_Permissions* in $\langle 7 \rangle$, which are particular to the instances of the components.

The output Parameterised RBAC (*PFRBAC* in $\langle 8 \rangle$) would be induced from the extension of *FRBAC* using the schema *Parameterize_RBAC*. The components of this model would be named in conformity with RBAC, with the convention that they would be prefixed by the capital letter P. As demonstrated earlier on, the set of parameterized roles in the Parameterised RBAC (*Proles* in $\langle 9 \rangle$) would contain both the parameterized roles of *FRBAC* ($ParamRoles \times M$), and the remaining roles of *FRBAC* that have not been parameterized ($Roles - ParamRoles$). The same reasoning applies to the derivation of principals (*PPrincipals* in $\langle 12 \rangle$) and derivation of objects (*Pobjects* in $\langle 17 \rangle$) of the parameterized model.

Parameterize_RBAC

$\langle 1 \rangle$ $\Delta FRBAC?$
$\langle 2 \rangle$ $M? : \mathbb{P}\, PARAMETER$
$\langle 3 \rangle$ $ParamRoles? : \mathbb{P}\, ROLE$
$\langle 4 \rangle$ $ParamPrincipals? : \mathbb{P}\, PRINCIPAL$
$\langle 5 \rangle$ $ParamObjects? : \mathbb{P}\, OBJECT$
$\langle 7 \rangle$ $New\_Permissions? : \mathbb{P}\, PPERMISSION$
$\langle 8 \rangle$ $PFRBAC!$

$\langle 9 \rangle$ $PRoles! = (Roles - ParamRoles?) \cup (ParamRoles? \times M)$
$\langle 10 \rangle$ $\mathrm{dom}\, PRoleAllocation = PPrincipals$
$\langle 11 \rangle$ $\mathrm{ran}\, PRoleAllocation = PRoles$
$\langle 12 \rangle$ $PPrincipals! = (Principals - ParamPrincipals?) \cup (ParamPrincipals? \times M)$
$\langle 13 \rangle$ $\mathrm{dom}\, PSubjectAssociation = PSubjects$
$\langle 14 \rangle$ $\mathrm{ran}\, PSubjectAssociation = PPrincipals$
$\langle 15 \rangle$ $\mathrm{dom}\, PSubjectRole = PSubjects$
$\langle 16 \rangle$ $\mathrm{ran}\, PSubjectRole = PRoles$
$\langle 17 \rangle$ $PObjects! = (Objects - ParamObjects?) \cup (ParamObjects? \times M)$
$\langle 18 \rangle$ $POperations! = Operations$
$\langle 19 \rangle$ $PTasks! = Tasks$
$\langle 20 \rangle$ $PPermissions! = ((Roles - ParamRoles?) \lhd Permissions) \cup ParamPermissions \cup New\_Permissions?$
$\langle 21 \rangle$ $ParamPermissions = \{(r(m), p) \bullet (r, p) \in Permissions \land r \in ParamRoles \land m \in M\}$
$\langle 22 \rangle$ $\mathrm{dom}\, New\_Permissions \subseteq (ParamRoles \times M)$

3.3 The Nesting Property of the Parameterised RBAC Model

This work presents a methodology for parameterizing RBAC components in order to deduce a Parameterised RBAC model. The resulting model, which is also an RBAC model, can be further parameterized in order to achieve an additional level of granularity. The components of the new model would now depend on two parameters as shown in Figure 2. This figure depicts a hierarchy of parameters data. In it, we use the tree notation whereby a filled square denotes a leaf and a circle denotes a node with children. This parameterization can be repeated successively, and in a nested way, as long as required to achieve the required access control granularity. In this way, new parameterized RBAC model would be devised following the parameters' hierarchy, as shown in Figure 2. However, there can be another way of nesting parameters; which is by using the cross product of the set of parameters M and the RBAC com-

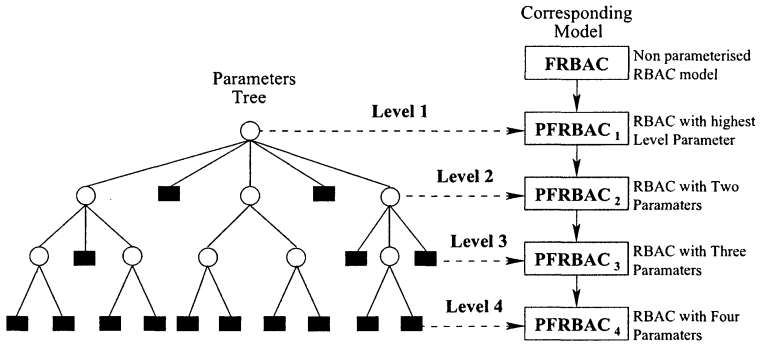

Figure 2. The nesting of the parameterized models following the level of the appropriate parameters.

ponents at every level. This means that we would parameterize all components at every level. In this type of nesting parameterization the order of parameters is not important. For instance, if in the online banking example both roles $Account\_Holder(branch(account_1))$ and $Account\_Holder(account_1(branch))$ are considered the same. The nesting by cross product results in a number of redundant parameterized RBAC components at every level, which might not be necessarily needed. This is prevented when using our methodology of parameterization because it enables the choice of parameters at every level of the parameters' tree. Also, it enables to parameterize only the required components such as roles, objects and principals. This reduces the number of parameterized components at every level and eases the manageability of the access rights.

4. Discussion

Two of the main advantages of RBAC are the simplification of access rights management and the presentation of a high level view of security in an organisation. However, in its current form, RBAC does not seem to have enough power to express a wide range of security requirements and capture fine access control granularity when put into application. These features can be accounted for by extending RBAC to the Parameterised RBAC, in order to support parameters, as shown earlier on in this work. The Parameterised RBAC provides finer granularity by creating instances of RBAC components according to the contexts of their use. With this, it can cater for special security requirements such as the support of private access rights for each of the instances of the same role, and the differentiation between the access rights of subjects associated with the same role. Providing very fine granularity can however complicate the access control list (ACL) because it involves handling access rights for a significantly

higher number of roles, which is due to the instantiation of the RBAC roles. This undermines the advantage of the simplicity of access rights management, for which RBAC is known. It seems there is a tradeoff between simplifying the management of access rights and providing fine granularity; and between providing a higher view of security, and parameterizing RBAC concepts to increase the expressive power of the Parameterised RBAC. Providing a balance between these advantages in a Parameterised RBAC model is greatly affected by the way the parameterization is performed. More specifically, this balance depends on two factors:

- The choice of the parameterized RBAC concepts: this involves deciding which RBAC concepts, such as *roles*, *objects* and *principals*, are to be parameterized.

- The type of parameters: the parameters according to which the parameterization would be performed are often related to the environment or context where the Parameterised RBAC is applied. As seen earlier on, parameters can be faculty names and unit names in a university, or a department name in a commercial organisation.

5. Conclusion

This paper has the proposed a rigorous formal model for the Parameterized RBAC (*PFRBAC*). One strength of this model lies in the methodology that is used to migrate from the direct implementation of an RBAC model, which suffers the drawbacks of inconsistency of access rights distribution, difficulty of intellectual manageability and the weak scalability of the model, into a parameterized RBAC which achieves the same results without bearing these drawbacks. This parameterized implementation of RBAC is very important for realistic applications because it achieves extremely fine grained access control granularity. Another strength of *PFRBAC* is its completeness in terms of investigating all the concepts and semantics of a Parameterised RBAC and supporting all the definitions and features of the well-known RBAC models in [1, 2, 4, 5]. The formalisation of *PFRBAC* in the Z notation makes it clear to understand and eliminates ambiguities at the application phase.

References

[1] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, Nov. 1996.

[2] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-Based Access Control: Towards A Unified Standard," in *Proc. of the 5th ACM workshop on Role-Based Access Control*. Technical University of Berlin, Berlin, Germany: ACM Press, June 2000, pp. 47–63.

[3] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, and R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[4] American National Standard for Information Technology, "*Role Based Access Control*," Draft BSR INCITS 359, Apr. 2003. Online: http://csrc.nist.gov/rbac/rbac-std-ncits.pdf.

[5] E. Khayat and A. Abdallah, "A Formal Model for Flat Role-Based Access Control," in *Proc. of the ACS/IEEE International Conference on Computer Systems and Applications*. Tunis, Tunisia: IEEE Press, July 2003.

[6] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2003.

[7] Sun Microsystems. *RBAC in the Solaris Operating Systems*. White Paper, April 2001. http://wwws.sun.com/software/whitepapers/wp-rbac/wp-rbac.pdf.

[8] T. Chalfant. *Role Based Access Control and Secure Shell–A Closer Look At Two Solaris™ Operating Environment Security Features*, June 2003. http://www.sun.com/solutions/blueprints/0603/817-3062.pdf.

[9] E. Lupu and M. Sloman, "Reconciling Role Based Management and Role Based Access Control," in *Proceedings of the 2nd ACM workshop on Role-based Access Control*. Fairfax, Virginia, USA: ACM Press, Nov. 1997, pp. 135–141.

[10] D. Gollmann, *Computer Security*. John Wiley & Sons, 1999.

[11] T. Jaeger, T. Michailidis, and R. Rada, "Access Control in a Virtual University," in *Proc. of the 8th International IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, California , USA, June 1999, pp. 135–140.

[12] L. Bottaci and J. Jones, *Formal Specification Using Z: A Modeling Approach*. International Thomson Computer Press, 1995.

[13] J. Bowen, *Formal Specification & Documentation Using Z: A Case Study Approach*. International Thomson Computer Press, 1996.

[14] I. Toyn (Ed.), "*Information Technology-Z Formal Specification Notation-Syntax, Type System and Semantics*," Consensus Working Draft 2.7, Oct. 2001.

[15] L. Giuri and P. Iglio, "Role Templates for Content-Based Access Control," in *Proc. of the 2nd ACM Workshop on Role-Based Access Control*. Fairfax, Virginia, USA: ACM Press, Nov. 1997, pp. 153–159.

[16] Jean Bacon, Ken Moody and Walt Yao. A model of OASIS role-based access control and its support for active security. *ACM Trans. Inf. Syst. Security*. 5(4): 492-540 (2002)

[17] Andras Belokosztolszki, David M. Eyers and Ken Moody. Policy Contexts: Controlling Information Flow in Parameterized RBAC. *POLICY 2003*: 99-110.