Towards Modal Logic Formalization of Role-Based Access Control with Object Classes

Junghwa Chae

École Polytechnique de Montréal
Montréal, Québec, Canada
chae@cse.concordia.ca

Abstract. This paper addresses a variation of the role-based access control (RBAC) model with a classification mechanism for objects and a notion of class hierarchies. In the proposed model, the authorization tasks are performed based on the classes instead of the individual objects. This results in more flexibility in terms of security administrative tasks such as downgrading or upgrading individual objects and permission assignments. A formalization for this model is presented using $K45$ modal logic. The prefixed tableaux method is used to reason about the access control. The required rules for the reasoning process are also presented. The proposed model is applied, via an example to protect the secrecy of the information in a typical organization.
Keywords: Role-based access control, object classes, object class hierarchy, modal logic, tableaux method.

1 Introduction

Role-based access control (RBAC) provides the abstraction mechanism for categorizing users in roles based on the organizational responsibilities of users [2, 6, 8, 13]. The role is the association between a set of users and a set of permissions. The role simplifies security management tasks to grant and revoke authorizations to an entire group of subjects at the same time. The defined roles can also have hierarchical structures for more convenient authority managements.

In our analysis of security, we provide the RBAC model with a classification mechanism for objects accessed in information systems. Our thesis is that objects are classified into groups called object classes, and classes can constitute a systematic structure, known as a hierarchy. Once objects are categorized into groups, authorization tasks can be executed based on the classes instead of the individual objects. Semantically or functionally related object classes associate with each other via inheritance relationships, and objects can be involved in these hierarchical relationships through the classes in which they are categorized. Object class hierarchy is a method to achieve further simplification in the reduction of security management tasks and administrative costs. It also provides a way to control the propagation of authorizations and to define boundaries for the validity of authorization rules. This modification of the RBAC model provides greater control and flexibility for the security administrative tasks.

Formal methods and reasoning techniques are useful tools for the representation and decision of access control. In this paper, we present a logical approach based on a modal logic formalism [1, 5, 9]. There already exists a well understood theory of how modalities interact with propositional logic connectives. This framework provides a language for expressing properties and relationships of security policies without considering the specific mechanisms for implementing such policies. The semantics of the policy definitions for security is provided using Kripke structures [5]. In developing a formalism for the proposed model, our main contribution is the incorporation of the notion of object classes into the work done by Abadi [1] and Massacci [9].

There has been much research done on logical frameworks for the reasoning of access control models. Woo and Lam [15] proposed a language to model authorization and control rules. A major issue in their approach was the trade-off between expressiveness and efficiency. For the logical formalism approach, Jajodia et al. [7] proposed a logic-based language for specifying authorization rules. Massacci [9] introduced a logic for reasoning about RBAC, by extending Abadi et al.'s access control calculus [1]. They used modal logic to model concepts such as users, roles, and delegation. Rabitti et al. [11] presented a model of authorization for next-generation database systems using the notion of implicit authorization. They developed an authorization model by including the properties of a class, class hierarchy, and composite objects. Bertino et al. [3] proposed a formal framework for reasoning about access control models. They introduced the concepts that subjects, objects, and privileges can be composed together in hierarchical structures and authorization can be derived along the hierarchies. Most existing work on RBAC concentrated on key points such as role hierarchies, user and privilege attributions. There has been little work that studied the role-object relationships and the hierarchy for object classes in RBAC. The idea of object classification for role-based policies was first introduced by Sandhu and Samarati [14]. An RBAC model that includes the concept of object classes was presented by Chae et al. [4] where the formalization was provided by description logics. In this paper, we use this model together with the calculus developed and extended by Abadi et al. [1] and Massacci [9] for the formalization. The existing formalization, which is based on the modal logics, is modified to include the notions of classification of objects and class hierarchy.

The rest of this paper is organized as follows: We begin with describing the proposed RBAC model with object classification in Section 2. The language developed by Abadi and extended by Massacci is reviewed in Section 3. The syntax required to support the notion of object classes and their hierarchy is given in the same section. The semantics for the existing and proposed operators will be discussed in Section 4. Rules for the reasoning process are presented in Section 5. The application of the proposed model and its formalization is illustrated within an example in Section 6. We summarize our results in Section 7 and conclude with suggestions for future work.

2 Role-Based Access Control

RBAC provides the abstraction of subjects based on the inherent properties of accesses. The abstraction of subjects organizes users with roles reflecting their real job functions or their responsibilities. This approach simplifies security management by breaking user authorizations into two parts: one which assigns users to roles and one which associates access rights to objects for those roles (see Fig. 1).

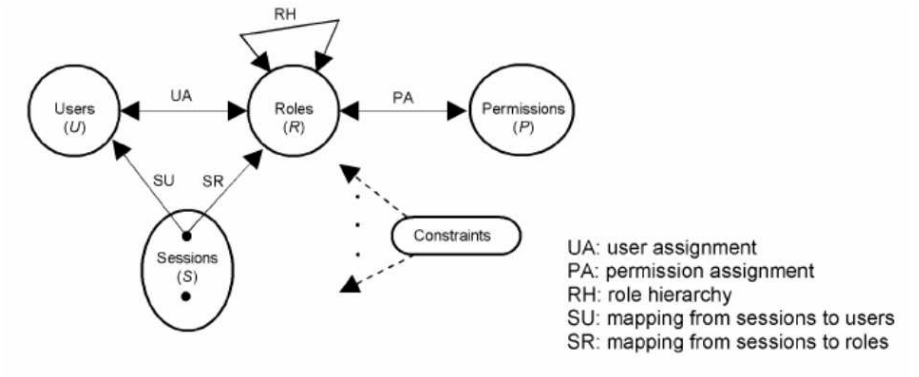

Fig. 1. RBAC model

Analogously, one might expect to achieve further simplification in the security management if some abstraction is provided for objects. Objects could be classified according to their type or to their application area. Grouping objects into classes closely resembles the mapping of users to roles. Fig. 2 shows the proposed model, which consists of five entities including a set of objects and a set of classes. We also added a set of object assignments (OA) that relates each object to a set of classes. Access authorizations of roles should then be defined based on the object classes. A role can be given the authorization to access all objects in a class, instead of giving explicit authorization for each individual object. Objects that are in the same class can be accessible for users with roles that have access right to that class. Ultimately, users exercise permissions on objects via roles to which they are assigned and classes to which the roles have access. We consider roles and object classes as mediators that let users exercise permission. The modified model decomposes each permission into an operation and an object. Therefore, the Permissions entity as depicted in Fig. 1 is removed and two new components Objects and Classes are added in Fig. 2.

This modification of the RBAC model provides greater control and flexibility for security administrative tasks. In particular, this approach simplifies and eases the authorization management ; e.g., in order to add a new object to the system, the corresponding object assignment assertion should only be included, whereas

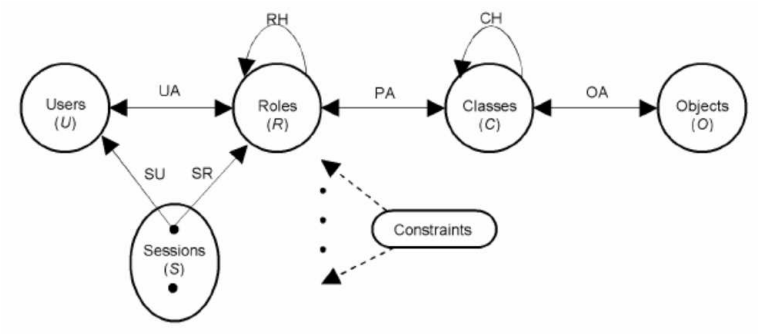

Fig. 2. Proposed modified RBAC model

in the RBAC model, permission assignment should be explicitly given for each single role that has the privilege of accessing the new object. Compared to roles, object classes have a greater potential for simplifying security administration since the number of objects in many systems is generally much larger than the number of subjects.

2.1 Role Inheritance

In RBAC, roles are hierarchically organized into a role-subrole relationship that is called *role inheritance*. The hierarchy is interpreted using a graph where each node represents a role and a directed edge between two roles defines the implication of the authorization. Authorizations are implied along the edges of the role hierarchy. When role R_1 inherits from role R_2, denoted by R_1 isa R_2, every user U explicitly assigned to R_1 is also implicitly associated with R_2; likewise, every permission explicitly associated with role R_2 is implicitly associated with role R_1.

The role hierarchy is a partial order relation, which is reflexive, transitive, and antisymmetric. Inheritance is reflexive because a role inherits its own permissions; transitivity is a natural requirement in this context, and antisymmetry rules out cycles in the role hierarchy; i.e., roles that inherit from one another are disallowed.

2.2 Class Inheritance

In the proposed model, a set of objects are grouped together for security purposes. Each group is, in general, a set of individual objects, and is referred to as a class. Objects are associated with certain properties that can be used to construct groups for the authorization process. Examples of object properties are security levels, ownerships, classes (as in the object-oriented terminology), memberships, etc. Once the objects are categorized into finite sets of groups,

authorization tasks can be executed based on the classes instead of individual objects.

Object classes are also organized into a hierarchical structure, called *class inheritance* (note that the word class here is not used in the sense of object-oriented programming but represents any named group of objects). The hierarchy can be based on different criteria such as security levels, generalization and specialization associations, as in object-oriented systems, and so on.

In the role inheritance, the concept of implied authorization is applied. The idea is to propagate the validity of the authorization rule at some level in a hierarchy to its descendants [11]. Similarly, the same idea can be applied to object classes through a hierarchy. Class hierarchies coupled with role hierarchies are implemented in the reasoning process. The definition of object classes and its hierarchical structure provides more reasoning power compared to the conventional RBAC approach.

We propose the following authorization policies:

- Access to a class implies access to the objects explicitly assigned to that class;
- The class hierarchy is defined as follows: the relation $C_1 \preceq_p C_2$ means that all roles given an access privilege p on class C_1 have the same access privilege on class C_2. Therefore, a user U who has a certain access to class C_1 is allowed to exercise the same access on class C_2. In general, the direction of the above inequality relation depends on the type of the operation; e.g., there may exist another operation denoted by p' for which the class inheritance relation between C_1 and C_2 would change to $C_2 \preceq_{p'} C_1$; for example, read and write operations in mandatory policies where classes are formed based on the security level (access classes). In this situation we can replicate each classes by the number of operations that have different hierarchical relations; e.g., C_1^p and $C_1^{p'}$.

3 Language for Access Control

In this paper, we adopted the calculus developed by Abadi [1] et al. to model access control in distributed systems. We equipped the calculus with proper notations to describe the proposed concept of object classes and their hierarchy in the context of modal logic.

The main syntactical components of this logic [1, 9, 10] are principals, requests and a set of modal and propositional connectives and operators. Users and roles are examples of atomic principals. Atomic users are denoted by A and B and roles by R. Composite principals denoted by P and Q are built by the use of different connectives such as the conjunction of principals (A & B), users in a certain role (A as B) and principal on behalf of another principal (A for B). A complete list of principals in typical distributed information systems is given by Abadi [1].

The common practice in RBAC is to represent the combination of an operation over an object as an atomic request or a statement; e.g., *read_file1*. In this

approach, neither the operation nor the object by itself is considered as part of the model; whereas the combination or the request is considered as a proposition that can be true or false depending on the state of the system. Composite requests are then built using propositional logic connectives $\wedge$, $\neg$, and $\Rightarrow$. Since our main objective is to categorize individual objects into object classes and to use the class hierarchy for a reasoning process, following this common practice would be inappropriate. In the operation-object approach, two statements such as *read_file1* and *write_file1* are considered as two independent propositions ϕ and ψ. However, our goal is to be able to separate these propositions into an operation part plus an object part. This distinction makes it possible to use the hierarchy associated with objects for reasoning about access control.

Atomic objects and classes are denoted by O and C, respectively. Object classification is made using the statement O belong C, which closely resembles Massacci's user assignments to roles, A has R [9]. belong and has are both modal operators. The operation over objects, read O or write O, is considered as an atomic request which is a proposition in the model and can take different truth values depending on the state of the system. These propositions are constructed as a combination of an operator (write or read) and an object or a class (O or C). User assignment statements as well as object classification statements are also simple propositions. Composite propositions are formed by combining the simple ones using propositional logic connectives.

In order to access objects in the system or to perform operations, users and roles (users in roles) make the corresponding requests. The main purpose of the access control policies is to determine whether these requests should be granted or not. The requests are made using the modal operator req [9]; e.g., A req ϕ where ϕ is a proposition such as read O. The statement A req ϕ is considered as a request or proposition by itself.

Privileges are given to users and roles using the control statement [1]. The proposition R control ϕ gives the permission on ϕ to role R; i.e.,

$$(R \text{ control } \phi) \ \wedge \ (R \text{ req } \phi) \ \Rightarrow \ \phi. \tag{1}$$

Role hierarchies are defined using the isa modal operator; e.g., the statement R_1 isa R_2 means that role R_1 has at least all of the privileges that are assigned to R_2. Operator $\sqsubseteq$ is used to define class hierarchies. The similar statement $C_1 \sqsubseteq C_2$ means that all users or roles that are given certain privileges on class C_1 have at least the same privileges on class C_2 (permissions given on class C_1 are valid for class C_2). The role or object class hierarchy statements are also considered as requests or propositions.

Our objective is to benefit from role and class hierarchies to reason about access control; e.g., to be able to perform the following operations,

$$R_2 \text{ control } \phi \ \wedge \ R_1 \text{ req } \phi \ \wedge R_1 \text{isa } R_2 \Rightarrow \phi,$$
$$R \text{ control } (\text{read } C_1) \wedge R \text{ req } (\text{read } O) \wedge O \text{ belongs } C_2 \wedge C_1 \sqsubseteq C_2 \Rightarrow \text{read } O.$$

4 Semantics

The semantics are defined using Kripke structures [5]. The syntax of the language described in the previous section consists of a set of agents (atomic principals), objects and classes, a set of primitive propositions Φ, atomic requests mainly of the form read O, and modal as well as propositional connectives and operators to construct primitive propositions from objects and classes or build composite agents and propositions. A Kripke structure denoted by $\mathcal{M}$ for a set of agents over Φ is a pair $(W, \mathcal{I})$ where W is a set of possible worlds (states) for a typical information system and $\mathcal{I}$ is the interpretation function. What makes each world distinct from the other is its specific truth values for the set of primitive propositions. Each agent (principal) A is interpreted as a set of pairs such that each pair consists of elements of W, i.e., $A^{\mathcal{I}} \subseteq W \times W$. Pair $(w_i, w_j) \in A^{\mathcal{I}}$ indicates that state w_j is one of the compatible states with state w_i for agent A. The interpretation of a proposition (request) is a set of states where the proposition (request) is true (granted), i.e., $\phi^{\mathcal{I}} \subseteq W$. Classes are interpreted as set of worlds in which they are accessible. Objects are uninterpreted entities within the structure.

4.1 Principals and Hierarchies

The interpretations of a user or a role is given by a set of pairs that define the compatible worlds (states) for the corresponding role or user; e.g.,

$$A^{\mathcal{I}} = \{(w_i, w_j), (w_i, w_k), (w_i, w_m), (w_j, w_n), (w_j, w_m), (w_j, w_p), \ldots\}, \quad (2)$$

where w_j, w_k, w_m are among the compatible states with state w_i for user A; i.e., the state of the system will change from w_i into one of the compatible states according to the requests made by user A in state w_i.

For the role hierarchy given by R_1 isa R_2, since R_1 can also act as R_2, all compatible worlds of R_2 should also be among the compatible worlds of R_1; i.e.,

$$R_2^{\mathcal{I}} \subseteq R_1^{\mathcal{I}}. \quad (3)$$

As mentioned in Section 3, role hierarchy statements are considered as requests or propositions in the system, hence R_1 isa R_2 will be interpreted as a set of worlds where Relation (3) is valid:

$$(R_1 \text{ isa } R_2)^{\mathcal{I}} = \left\{w \middle| \forall w' \text{ if } (w, w') \in R_2^{\mathcal{I}} \text{ then } (w, w') \in R_1^{\mathcal{I}}\right\}. \quad (4)$$

4.2 Object Classes and Hierarchies

Each object class is interpreted as a set of states where it is accessible. As explained in Section 2, when there are different types of operations in the information system, we replicate each class by the number of operations, e.g., C^r

and C^w. In this case the interpretation of C^r or C^w indicates the set of worlds where the class can be read or written, respectively,

$$C^{r[w]^{\mathcal{I}}} = \{w_i, w_j, w_k, \ldots\}. \tag{5}$$

The state w_i exists in the interpretation of the object class $C^{r[w]}$ iff the statement $\mathsf{read}[\mathsf{write}]\ C^{r[w]}$ is valid in this state.

As seen in Section 3, the statement $C_1 \sqsubseteq C_2$ indicates that all permissions given on class C_1 are also valid for class C_2. In all states where class C_1 is accessible, class C_2 is accessible too. From Relation (5), it follows that all states in the interpretation of class C_1 should exist in the interpretation of class C_2; i.e.,

$$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}. \tag{6}$$

The class hierarchy statement can then be interpreted as,

$$(C_1 \sqsubseteq C_2)^{\mathcal{I}} = \left\{w |\text{ if } w \in C_1^{\mathcal{I}} \text{ then } w \in C_2^{\mathcal{I}}\right\}. \tag{7}$$

4.3 Request Operator and Properties

The definition of the request operator req is similar to Fagin et al.'s knowledge operator [5]. It has two arguments: a principal that makes the request and a proposition that is requested. A request statement made in state w by principal A is valid when its propositional argument is true in all compatible states with w,

$$(\mathcal{M}, w) \models A\ \mathsf{req}\ \phi \text{ iff } (\mathcal{M}, w') \models \phi \text{ for all } w' \text{ such that } (w, w') \in A^{\mathcal{I}}. \tag{8}$$

Compatible states with state w for principal A are defined as a set of states where all of the requests made by A in w will be granted.

The truth of $A\ \mathsf{req}\ \phi$ does not imply that ϕ is granted. In fact, the access control system has the responsibility of verifying whether ϕ should be granted whenever $A\ \mathsf{req}\ \phi$ is valid, i.e., the process of verifying if ϕ is true (granted) starts after it is proven that the $A\ \mathsf{req}\ \phi$ is true. The distinction between granting a request and the truth of a request statement is crucial. According to the definition (8), if ϕ is invalid in any of the compatible states with w for principal A, then A is unable to make a valid request in w, i.e., $(\mathcal{M}, w) \not\models A\ \mathsf{req}\ \phi$. Whereas, when ϕ is true (granted) in all compatible states with w, then the request statement $A\ \mathsf{req}\ \phi$ is valid in w and the reasoner starts the process of verifying whether the propositional argument of the request statement should be granted.

The binary relations formed by the interpretation of users or roles exhibit certain properties. Relations satisfying $K45$ properties; i.e., transitive and Euclidean, fit best with the characteristics of access control in information systems. The Euclidean property requires that for all $w_1, w_2, w_3 \in W$ if $(w_1, w_2) \in A^{\mathcal{I}}$ and $(w_1, w_3) \in A^{\mathcal{I}}$, then $(w_2, w_3) \in A^{\mathcal{I}}$. The interpretation relations for principals should not be reflexive as the reflexivity necessitates that all valid requests

made in state w should be granted in the same state. The transitivity is required since successive requests can be combined into one composite request. Similarly, a composite request made in state w can be considered as subsequent individual requests that necessitates Euclidean property for the binary relations. Given the definition (8) as well as the transitive and Euclidean binary relations for users and roles, the request operator holds the following properties in the Kripke structure $\mathcal{M}$:

K $\vdash (A \ \mathsf{req} \ \phi \ \wedge \ A \ \mathsf{req} \ (\phi \Rightarrow \psi)) \ \Rightarrow \vdash A \ \mathsf{req} \ \psi$;
KG if $\vdash \phi$ then $\vdash A \ \mathsf{req} \ \phi$;
4 $\vdash A \ \mathsf{req} \ \phi \ \Rightarrow \vdash A \ \mathsf{req} \ (A \ \mathsf{req} \ \phi)$;
5 $\vdash \neg A \ \mathsf{req} \ \phi \ \Rightarrow \vdash A \ \mathsf{req} \ \neg (A \ \mathsf{req} \ \phi)$;
Id $\vdash A \ \mathsf{req} \ (A \ \mathsf{req} \ \phi) \ \Rightarrow \vdash A \ \mathsf{req} \ \phi$.

K property is the direct consequence of the definition of the request operator. It is valid whether or not binary relations exhibit transitive or Euclidean property. **K** rule indicates that when a user makes a request, it also includes all the logical consequences of her original request. In knowledge representation, properties **4** and **5** are called positive and negative introspection axioms, respectively. The former follows from the transitive property of the binary relations and the latter is the result of both transitive and Euclidean properties. **KG** is the knowledge generalization rule that says if ϕ is granted in all states of structure $\mathcal{M}$, then A *req* ϕ is true everywhere. **Id** is the result of Euclidean property. Properties **Id** and **4** show the idempotence of the req operator, i.e., the following equivalence relation is valid,

$$A \ \mathsf{req} \ (A \ \mathsf{req} \ \phi) \equiv A \ \mathsf{req} \ \phi. \tag{9}$$

4.4 User Assignment and Object Classification

The interpretation of user assignment statements has operator is given by,

$$(A \ \mathsf{has} \ R)^{\mathcal{I}} = \left\{ w | \, \forall w' \text{ if } (w, w') \in R^{\mathcal{I}} \text{ then } (w, w') \in A^{\mathcal{I}} \right\}. \tag{10}$$

The object classification statement $O \ \mathsf{belong} \ C$ is interpreted as follows:

$$O \ \mathsf{belong} \ C^{r[w]} \ \Rightarrow \ \mathsf{read}[\mathsf{write}] \ O \equiv \mathsf{read}[\mathsf{write}] \ C^{r[w]}. \tag{11}$$

4.5 Read and Write Statements

In Section 3, the read and write operations on objects and classes are constructed with read and write operators, respectively. From the interpretation of object classes given in Section 4.2, it follows that the statement $\mathsf{read} \ C^r$ is true in world w iff $w \in (C^r)^{\mathcal{I}}$. Hence read and write statements, only for object classes, are interpreted below,

$$\left(\mathsf{read}[\mathsf{write}] \ C^{r[w]} \right)^{\mathcal{I}} = \left\{ w | \, w \in C^{r[w] \, \mathcal{I}} \right\}. \tag{12}$$

Object interpretation is the similar to the class interpretation, i.e., it is given by a set of individual states. However, the interpretation of objects do not remain constant even within a single state. Depending on the requests made by principals to read or write an object in state w, its interpretation changes to either $C^{r\,\mathcal{I}}$ or $C^{w\,\mathcal{I}}$, respectively. Relation (11) is used to convert read and write operations on objects to the read and write operations on classes to which they belong upon the corresponding requests that are made.

5 Rules and Reasoning

Although the semantic is given by the Kripke structure, reasoning at the level of the structure would be inconvenient in access control systems. A set of inference rules are then introduced. These rules together with the axioms form an axiom system. Axioms are mainly given by the access control security policy as the ACL (Access Control List), which consists of the following statements:

- Role hierarchies (RH), $\bigwedge_{i,j} (R_i \ \mathsf{isa}\ R_j)$;
- Object class hierarchies (CH), $\bigwedge_{i,j} (C_i \sqsubseteq C_j)$;
- User assignments (UA), $\bigwedge_{i,j} (A_i \ \mathsf{has}\ R_j)$;
- Object classifications (OC), $\bigwedge_{i,j} (O_i \ \mathsf{belong}\ C_j)$;
- Permission assignments (PA), $\bigwedge_{i,j,k} \left(P_i \ \mathsf{control}\ \mathsf{Op}_j\, C_k\right)$; $\mathsf{Op}_j \in \{\mathsf{read}, \mathsf{write}, \ldots\}$.

The proof method is based on Massacci's prefixed tableaux algorithm [10]. This method is used to test the satisfiability of a proposition. The tableaux method builds a tree-like model $\mathcal{M}$ based on the input proposition and the global axioms. In tree $\mathcal{T}$, each node is labeled with a proposition and has a prefix that indicates the current state of the system. Tableaux rules are then repeatedly applied to nodes in an arbitrary order for as long as possible. A branch $\mathcal{B}$ of tree $\mathcal{T}$ is fully expanded when all rules have been applied to the nodes in $\mathcal{B}$. There exists a clash in $\mathcal{B}$ if a proposition and its negation exist in $\mathcal{B}$ with the same prefix. The proposition ϕ is valid in an axiom system built based on a set of global axioms G, if all branches of tree $\mathcal{T}$ that start with $\neg\phi$ lead to clashes. This indicates that $\neg\phi$ is not satisfiable.

The rules for $K45$ modal logic, users in roles and role hierarchies are due to Massacci [9] and are shown in Fig. 3. Here, σ is the current state of the system. $\sigma.A.n$ and $\sigma.A.m$ are present and new compatible states with σ according to the requests of principal A, respectively. Fig. 4 shows the required rules for object class hierarchies where $\mathsf{Op}_j \in \{\mathsf{read}, \mathsf{write}, \ldots\}$.

6 Example: RBAC Policies with Object Class Hierarchies

In this section, we illustrate a simplified model of a company with marketing and R&D departments. The model includes six different roles: Administrator, R&D-manager, R&D-staff, Marketing-manager, Marketing-staff, Customer. The role

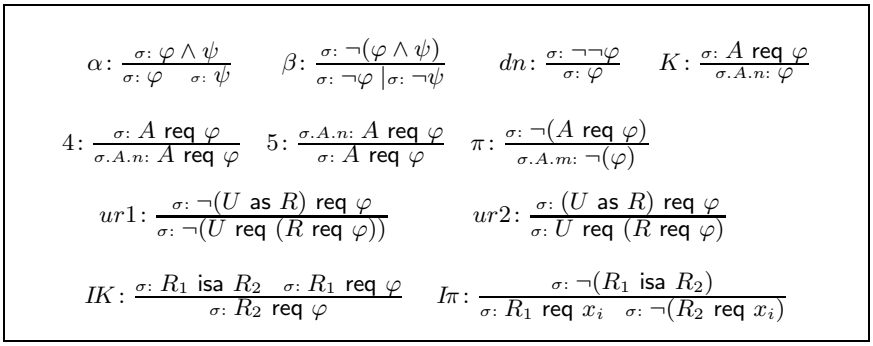

Fig. 3. Tableaux rules based on $K45$ properties, users in roles, and role hierarchy

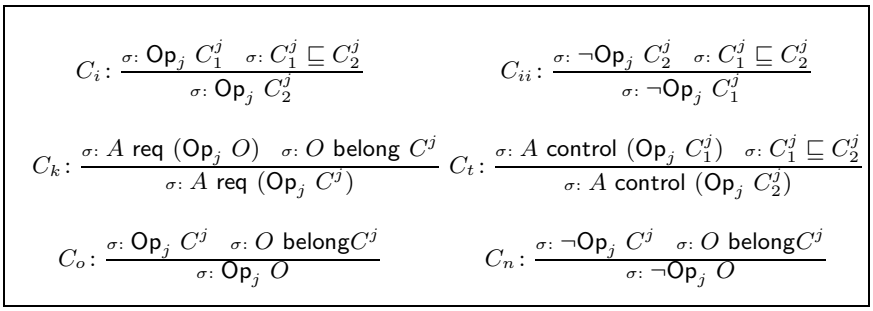

Fig. 4. Rules for object classes and hierarchies

hierarchy shown in Fig. 6 is derived based on the lattice $\mathcal{L}$ of the access classes depicted in Fig. 5. Access classes are composed of a set of category and a security level [12]. In this example the set of categories are subsets of {Marketing, R&D} and security levels are defined by C (classified) and U (unclassified). We use the following abbreviation to represent the concept of roles: Admin, RDMag, RDStf, MktMag, MktStf, Cust. The role hierarchy (RH) is modeled using the following inclusion axioms:

Admin isa RDMag, Admin isa MktMag, RDMag isa RDStf,
MktMag isa MktStf, RDStf isa Cust, MktStf isa Cust.

Objects are classified into six categories: Company-agenda, Patent, Technical-report, Contract, Marketing-survey, and General-information. Two object classes are defined for each category; one for read and the other for write operation. Object class concepts are defined as: Agenda^r, Patent^r, TechRep^r, Contract^r, MktSur^r, Geninfo^r, Agenda^w, Patent^w, TechRep^w, Contract^w, MktSur^w, Geninfo^w. The classes for read and write are distinguished by superscripts r and w, respectively. The class hierarchy is shown in Fig. 7. This hierarchy is also derived from

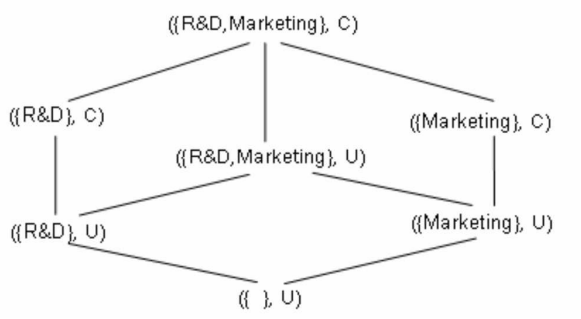

Fig. 5. Security lattice $\mathcal{L}$

access classes in Fig. 5. The inheritance relations among classes are given by the following class hierarchy axioms (CH):

$$\begin{array}{l}\text{Agenda}^r \sqsubseteq \text{Patent}^r,\ \text{Agenda}^r \sqsubseteq \text{Contract}^r,\ \text{Patent}^r \sqsubseteq \text{TechRep}^r,\\ \text{TechRep}^r \sqsubseteq \text{Geninfo}^r,\ \text{Contract}^r \sqsubseteq \text{MktSur}^r,\ \text{MktSur}^r \sqsubseteq \text{Geninfo}^r,\\ \text{Geninfo}^w \sqsubseteq \text{TechRep}^w,\ \text{TechRep}^w \sqsubseteq \text{Patent}^w,\ \text{Patent}^w \sqsubseteq \text{Agenda}^w,\\ \text{Geninfo}^w \sqsubseteq \text{MktSur}^w,\ \text{MktSur}^w \sqsubseteq \text{Contract}^w,\ \text{Contract}^w \sqsubseteq \text{Agenda}^w.\end{array}$$

Permissions are assigned such that they relate roles and object classes that

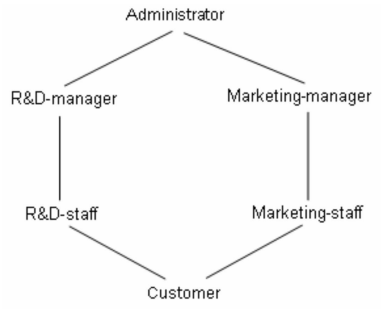

Fig. 6. Role hierarchy

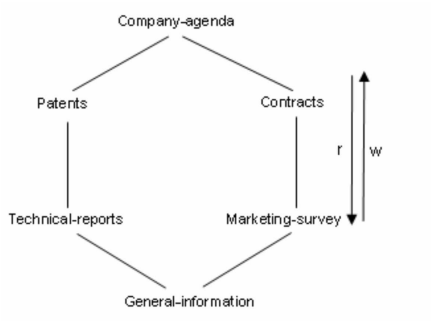

Fig. 7. Class hierarchy

are located at the same level in the hierarchy; e.g., RDMag can read and write Patent^r and Patent^w, respectively. Permission assignment axioms (PA) for all

roles that exist in the model are shown below.

Admin control (read Agenda^r), Admin control (write Agenda^w),
RDMag control (read Contract^r), RDMag control (write Contract^w),
RDStf control (read TechRep^r), RDStf control (write TechRep^w),
MktMag control (read Contract^r), MktMag control (write Contract^w),
MktStf control (read MktSur^r), MktMag control (write MktSur^w),
Cust control (read Geninfo^r), Cust control (write Geninfo^w).

The class hierarchy reduces the number of permission assignment axioms; e.g., for Admin, it is sufficient to specify the read permission only over the class Agenda^r. All read permissions over other classes for Admin can be implied using the class hierarchy. A similar inference capability based on the role hierarchy already exists in the RBAC model, e.g., the specification of permissions for RDStf implicitly gives the same permissions to RDMag. However, the class hierarchy provides additional axioms that can be used together with the role hierarchy to enhance the reasoning power.

Suppose user Bob who is assigned to role Marketing-manager wishes to read file $f1$ that is classified under Marketing-survey, MktSur^r. This request is equivalent to the following relation:

$$\text{Bob req read } f1.$$

We prove that the negation of the propositional argument of the above request, $\neg$read $f1$, is not satisfiable in any model that is built based on the global axioms RH, CH, UA, OC, and PA. The reasoning process (shown in Fig. 8) starts with the negation statement. The global axioms used in the process of reasoning are indicated with an italicized font. All branches that are shown here lead to clashes. Since the negation is not satisfiable; i.e., the relation itself is valid and Bob's request should be granted.

7 Conclusion and Future Work

The notion of object classes is appended to RBAC. A method is introduced, based on the modal logic, to formalize RBAC policies with object classes and to use object class hierarchies for reasoning about access control.

In our approach, we replicated classes by the number of operations (C^r and C^w), and interpreted each of them as a set of individual states, Relation (5). This approach closely resembles the original operation-object (*read_file1*) definition of permissions in RBAC. Whereas, using the proposed method, one only needs to replicate object classes rather than all individual objects. This results in a great simplification for typical information systems where the number of objects is usually quite large, however they can be categorized into few classes.

Not all axioms in the ACL have the same level of importance in an information system. While user assignment (UA) and object classification (OC) statements can be specified by local managers, role and object class hierarchies

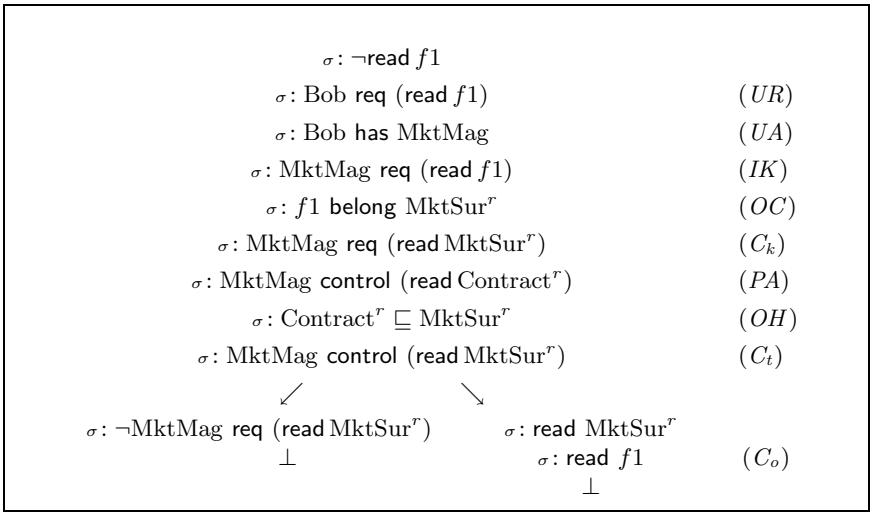

Fig. 8. Proving steps for the request *Bob* req read $f1$

as well as permission assignments can be considered as the signature of the access control policies and should be determined by high authority administrators. In the proposed method, object classification and class hierarchies are specified via different operators belong and $\sqsubseteq$. Operator belong provides a mechanism to easily downgrade (sanitize) or upgrade a specific object into the appropriate class, while $\sqsubseteq$ provides the hierarchies between classes determined by the security administrator.

The use of an expressive logic, such as modal logic, simplifies the application of different constraints that will be investigated in future work.

Acknowledgments. This research was supported by Institute for Information Technology Advancement (IITA) & Ministry of Information and Communication (MIC), Republic of Korea.

References

1. M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst. (USA)*, 15(4):706 – 734, Sept. 1993.
2. J.F. Barkely, V. Cincotta, D.F. Ferraiolo, S. Garrvrilla, and D.R. Kuhn. Role based access control for the world wide web. *NIST 20th National Computer Security Conference*, pages 331 – 340, 1997.
3. E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur. (USA)*, 6(1):71 – 127, 2003.

4. J.H. Chae and N. Shiri. Formalization of RBAC policy with object class hierarchy. In *Proc. of the 3rd Information Security Practice and Experience Conference (ISPEC)*, 2007.
5. R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. The MIT Press, Cambridge, Massachusetts, 1995.
6. D.F. Ferraiolo, J.F. Barkely, and D.R. Kuhn. A role based access control model and reference implementation within a corporate Intranet. *ACM Trans. Inf. Syst. Secur. (USA)*, 1(2):34 – 64, 1999.
7. S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst. (USA)*, 26(2):214 – 260, 2001.
8. M. Koch, L.V. Mancini, and F. Parisi-Presicce. A graph-based formalism for RBAC. *ACM Trans. Inf. Syst. Secur. (USA)*, 5(3):332 – 365, 2002.
9. F. Massacci. Reasoning about security: A logic and a decision method for role-based access control. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 1244:421 – 435, 1997.
10. F. Massacci. Tableaux methods for access control in distributed systems. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, 1227:246 –, 1997.
11. F. Rabitti, E. Bertino, Won Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM Trans. Database Syst. (USA)*, 16(1):88 – 131, 1991.
12. P. Samarati and S.C. Vimercati. *Foundations of Security Analysis and Design: Tutorial Lectures*, chapter Access Control: Policies, Models, and Mechanisms, pages 137–196. Springer Berlin Heidelberg, 2001.
13. 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.
14. R.S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40 – 48, 1994.
15. T.Y.C. Woo and S.S. Lam. Authorization in distributed systems: a new approach. *J. Comput. Secur. (Netherlands)*, 2(2-3):107 – 136, 1993.