Supporting Relationships in Access Control Using Role Based Access Control

John Barkley
National Institute of Standards and Technology
jbarkley@nist.gov

Konstantin Beznosov
Baptist Health Systems of South Florida
beznosov@baptisthealth.net

Jinny Uppal
Florida International University
juppal01@cs.fiu.edu

July 29, 1999

Abstract

The Role Based Access Control (RBAC) model and mechanism have proven to be useful and effective. This is clear from the many RBAC implementations in commercial products. However, there are many common examples where access decisions must include other factors, in particular, relationships between entities, such as, the user, the object to be accessed, and the subject of the information contained within the object. Such relationships are often not efficiently represented using traditional static security attributes centrally administered. Furthermore, the extension of RBAC models to include relationships obscures the fundamental RBAC metaphor.

This paper furthers the concept of relationships for use in access control, and it shows how relationships can be supported in role based access decisions by using the Object Management Group's (OMG) Resource Access Decision facility (RAD). This facility allows relationship information, which can dynamically change as part of normal application processing, to be used in access decisions by applications. By using RAD, the access decision logic is separate from application logic. In addition, RAD allows access decision logic from different models to be combined into a single access decision. Each access control model is thus able to retain its metaphor.

1 Introduction

The Role Based Access Control model and mechanism have proven to be useful and effective. This is clear from the many RBAC implementations in commercial products,[1] e.g., Oracle, Sybase, Lotus Notes, Microsoft Transaction Server. However, there are many common examples where access decisions must include other factors, e.g., user attributes (other than security attributes), object attributes, user relationships to other entities, time of day, location of user, in addition to roles in order to obtain a result of an authorization decision[1]. Consider access policies associated with content rating systems in entertainment media. A movie rated "PG13" may only be viewed by a minor over the age of 13 unless accompanied by an adult[2]. An access decision for this policy includes the following factors:

- the age attribute of the user,
- the rating attribute of the object, i.e., the movie to which access is sought,
- the relationship, i.e., simultaneous access, with another user, and
- the age attribute of the other user.

As illustrated in this example, relationships between entities associated with an access decision are often very important.

[1] Because of the nature of this paper, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available.

Such relationships, critical to an access decision, can include relationships between a user and the “owner” of information, a user and the “provider” of information, and/or the user and the “subject” of information. In some cases, roles can be used to represent relationships. However, using roles to express relationships may be inefficient and/or counter intuitive. When roles cannot be used to represent relationships, it is common to program access decision logic directly into an application. This approach restricts the ability to change access policy in a timely manner.

This paper further develops the concept of relationships for use in access control. It shows how relationships can be supported in access control decisions. We describe an approach for supporting relationships which uses RBAC in conjunction with the Object Management Group (OMG) specification for the Resource Access Decision facility (RAD)[3]. Using RAD to augment role based access decisions with other information retains the fundamental metaphor of the RBAC model. The RBAC model need not be changed by adding additional elements which may be part of other models. In addition, RAD enables the inclusion of other access control models within an access control decision. These other access control mechanisms may be elements of legacy systems whose use is required under certain circumstances.

This paper primarily uses examples from the healthcare domain which result from requirements placed on access to healthcare information by providers, as well as, local, state, and Federal government. These requirements include relationships. However, the approach described in this paper can equally apply in any domain which requires relationships in access decisions.

The paper is organized as follows: Section 1 is this Introduction; Section 2 provides a brief overview of RBAC; Section 3 introduces the relationship concept; Section 4 describes how relationships can be used in access control; Section 5 describes the OMG’s RAD facility; Section 6 gives an example of using RAD to implement an access policy in healthcare; Section 7 compares RAD to other approaches to enabling RBAC to make use of relationship information; Section 8 summarizes the results of the paper.

2 Overview of RBAC

Role based access control (RBAC) is an alternative to traditional discretionary (DAC) and mandatory access control (MAC). RBAC is capable of expressing policies particularly suited for commercial applications. A principle motivation behind RBAC is the ability to specify and enforce enterprise-specific security policies in a way that maps naturally to an organization's structure. Even a very simple RBAC model affords an administrator the opportunity to express an access control policy in terms of the way that the organization is viewed, i.e., in terms of the roles that individuals play in order to carry out the goals of the organization. With RBAC, it is not necessary to translate a natural organizational view into another view in order to accommodate an access control mechanism. The natural organizational view is the access control mechanism. As such, RBAC can be described as a form of non-discretionary access control in the sense that users are unavoidably constrained by the organization's protection policies.

In order to use RBAC, administrators use roles to describe the functions of individuals within the organization. The roles are treated as an attribute of an individual and typically represented as a character string. Based on the responsibilities required of individuals assigned these roles, the access requirements of these roles are determined and the appropriate permissions necessary for access are associated with each role. This RBAC description of the organization, i.e., in terms of roles and associated privileges, remains relatively fixed as individuals join/leave the organization, or change roles within the organization. Using the RBAC description, administrators assign individuals the permissions needed to do their jobs by assigning to individuals the roles associated with their jobs. The role assignment effectively enables the permissions by means of the RBAC mechanism.

Within most organizations, the fact that user/role associations change more frequently than role/permission associations results in reduced administrative costs when using RBAC as compared to associating users directly with permissions. It can be shown that the cost of administering RBAC is a factor of U+P while the cost of associating users directly with permissions is a factor of U*P where U is the number of individuals in a role and P is the number of permissions required to perform that role[4].

Sandhu[5] provides a characterization of RBAC models as follows:

1. RBAC0: the basic model with users associated with roles and roles associated with permissions.
2. RBAC1: RBAC0 with role hierarchies.
3. RBAC2: RBAC1 with constraints on user/role, role/role, and/or role/permission associations.

The use of hierarchies is common in models and implementations, and the use of constraints is becoming more common[6]. Other factors, such as, relationships, time, and location may be a required part of an access decision. When used as part of an access decision, these factors may or may not be included within the RBAC model. As a result of the natural organizational view provided by the RBAC metaphor and RBAC’s lower administrative costs, it is advantageous to use RBAC as the focus of an access decision.

3 Overview of Relationships

A typical software system models real world entities. Entities do not exist in isolation[7], and as such, a software system must also capture the relationships between entities.

Following are some examples of real world entities and their relationships:

- A person *owns* a house; a house is *owned by* one or more persons.
- A person is *employed by* one or more organizations; an organization *employs* one or more persons.
- A patient is *attended* by an attending physician (i.e., the attending physician is currently providing treatment to the patient), an attending physician *attends* one or many patients.

In many systems, relationships are modeled implicitly by capturing information about the relationship in one or more of the entities involved in the relationship. In fact, modeling languages like UML[8] allow different types of relationships to be captured implicitly (as an attribute in one or more of the entities involved)[2].

However, in some application domains, there is a need to define relationships between entities as explicitly as the entity itself. This may happen if the relationship has information that does not belong to either of the related entities. Consider the relationship between a person having an “employed by” relationship with an organization. This relationship may have certain information, like start date and employment policies, that cannot be stored by either the person (who may be employed by many organizations or who may be unemployed), or the organization (which may have other information like organization demographics). In such applications, there is a need to define an entity that captures information that belongs to the relationship. For such cases, defining the relationship as an entity can accomplish this.

Another class of applications we consider are those where relationships between entities are highly dynamic and change frequently. For example, a secretary often works for more than one person who permit their secretary to have read access to their calendar. If another secretary substitutes for the regular one, read access must be given to the substitute, i.e., to the person who has the relationship “employee’s secretary.”

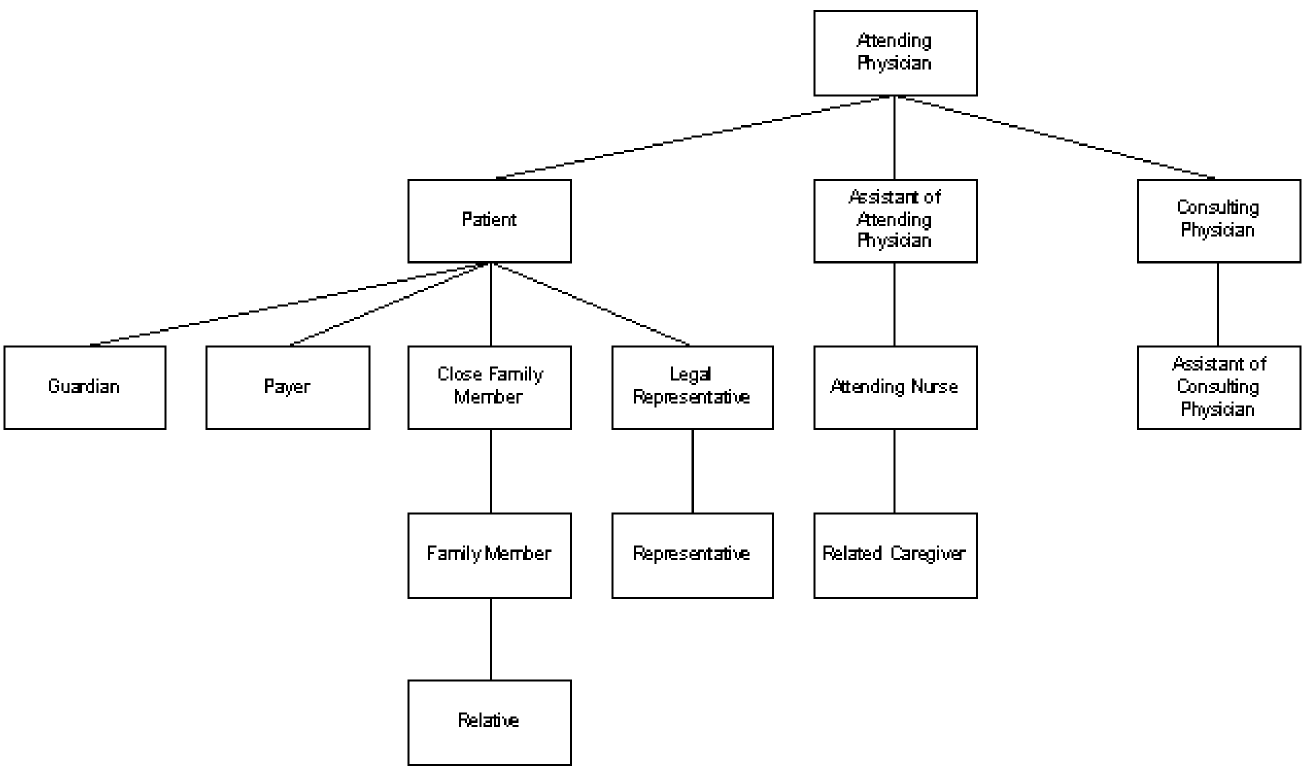

Figure 1. Sample relationship hierarchy for a patient under care.

Consider the relationships that can occur during interactions between a patient and a healthcare provider. Figure 1 illustrates the hierarchy of relationships that may be

[2] Relationships, in the context of this discussion, are also referred as ‘associations’ in UML.

referenced within an access policy for patient information. Such a hierarchy can have the same semantics of a hierarchy in RBAC, i.e., a parent node inherits all of the permissions of the child node. Note that this hypothetical hierarchy should be considered only in the context of privileges required to access patient medical data. It is shown as an example and it is not intended to be complete or necessarily fully correct.

Over a period of time, a patient establishes relationships with various other entities associated with the care provider. A patient referred to a care provider has a referral and a primary care physician. If admitted for care, the patient also has an admitting physician. In healthcare, relationships with physicians can be very important for enforcing access control policies. When a healthcare provider (the "user") accesses patient information (the patient is the "subject" of the information), the access policy may require information about the relationship between the provider and the patient. For example, section 3025, "Patient and Personnel Records; Copies; Examination" of Florida Hospital Licensing and Regulation[9] states that patient records must not be disclosed without consent of the patient except some cases. Included in the exceptional cases is the policy: "licensed facility personnel and attending physicians for use in connection with the treatment of the patient."

4 Relationships in Access Control

Relationships associate two or more entities. Relationships may be simple or complex. A simple relationship is something like a UML association, that is, it is stored as a pointer in either or all of the related objects. However, relationships may be complex depending on the amount of information that needs to be stored with them. For example, an employment object (that may relate a company object and a employee object) may store information like term of employment, salary etc. Lupu[10] defines roles and relationships as objects in accordance with an Object Model. Lupu also defines relationships to store policies regarding the related roles. For the purpose of this paper, the structure of the relationship is unimportant. Access decision policies may represent roles and relationships in any form.

RBAC introduces the concept of roles. With RBAC, users are assigned to roles, and permissions are also assigned to roles. This model helps make authorization decisions in systems where permissions are determined by the role of the user. Significant work has been done on relationships between roles by Lupu[10]. This paper discusses the use of relationships in role based access decisions where the relationships may exist between any two entities.

For example, in systems like healthcare, access decisions may also require information about relationships. Consider the relationship "attending-physician" between a physician and a patient. Most healthcare systems identify individual physician entities and maintain information about them, e.g., name, address, specialty. A physician typically has several attending-physician relationships with patients, just as a patient may have several attending-physicians. This relationship can be very dynamic in nature. An authorization service may need to know if an attending-physician relationship exists between patient Peter and the user with ID *smith*.

In an RBAC system where a role is a data attribute of the user, this could be modeled by creating an "attending-physician-to-Peter" role. Such a "fine-grained" role could be used to represent the relationship. However, this approach, when used in large systems with thousands of such relationships created and deleted dynamically, could create an explosion of roles, making management of the roles expensive and error-prone. In particular, this approach can result in:

- the attending-physician relationship likely being redundantly stored in two locations: the patient record database and the role database;
- security administrators having to update the role database whenever the attending-physician relationship changes for a user, and;
- potentially very large active role sets when the user is a doctor.

The alternative approach would be to include relationship information in the access decision. Permissions for a user in the physician role would be determined based on the physician role, the individual patient, and the attending-physician relationship between the user in the physician role and the individual patient.

It should be noted that relationships, as opposed to traditional security attributes such as roles, can have short lifetimes. As such, they may not be maintained by a central authority such as a security administrator. Instead the relationships will likely be managed in other components of the system, e.g., the attending-physician relationship in the healthcare example may be managed by the registration component of the healthcare information system.

Confronted with access policies which include relationships, the RAD facility is one approach for including relationship information in role based access decisions. The next section introduces the RAD facility.

5 Resource Access Decision Facility

The Resource Access Decision facility (RAD)[3] is an authorization framework and interface specification for distributed processing environments. The RAD facility was proposed in response to the Healthcare Resource Access

Control RFP issued by the OMG. RAD has the following major features:

- It enables the separation of application logic from authorization logic, hence providing a logically single point of administrative reference monitoring disparate application systems.
- It enables authorization decisions for resources of any nature and granularity as long as those resources can be named according to RAD's resource naming scheme.
- It enables the use of more than one authorization engine for decisions about the same request or different requests. These engines can support different authorization policies, can be integrated with legacy systems, and/or can be independently managed by independent authorities.
- It enables use of such request-specific or user-specific factors in authorization decisions that:
  - can change values during user session, i.e., the most current values be obtained when an access decision is required,
  - may be part of the request context,
  - can have values set as part of normal business processes as opposed to being set by a security administrator.

5.1 Interaction Between Application Service and Authorization Service

The main objective of RAD is to separate authorization logic from application logic. Authorization logic is encapsulated into an authorization service external to the application. A simplified schema of interactions among the application client, an application service, and an instance of the RAD is depicted in Figure 2. To perform application-level access control, an application obtains an access decision from such a service and enforces that decision. Simple interfaces between the application and the access decision service are used. An application programmer need only make a single invocation on the access decision service in order to obtain a decision.

The sequence of the interaction, illustrated by Figure 2, is as follows:

1. An application client invokes an operation on the application service (application, for short).
2. While processing the invocation, the application obtains an access decision from the RAD.
3. The RAD makes an authorization decision, which is returned to the application.
4. The application, after receiving the access decision, enforces it. If access was granted by the RAD, the application returns expected results of the invocation. Otherwise, it either returns partial results or denies access.

An application obtains an access decision from only one instance of RAD. It is the contract between the application and its enterprise environment to request an authorization decision and to enforce it. Before we proceed with greater details on the design of an access decision service, we will describe the syntax and semantics of an access decision request.

From the perspective of RAD, any application requesting an authorization decision is a RAD client. From now on, we will use the term "RAD client'" to refer to any enterprise entity of the system that requested an authorization decision from a RAD.

A nominal amount of data is passed between the application and the access service in order to make authorization decisions. When making a request for an authorization decision, a RAD client passes the following three parameters:

- a sequence of name-value pairs representing a name of the resource to be accessed on behalf of the client,
- name of the access operation (e.g. "create", "read", "write", "use", "delete"),
- the security attributes of the authenticated subject (i.e., the user) on behalf of which the client is requesting access to the named resource.

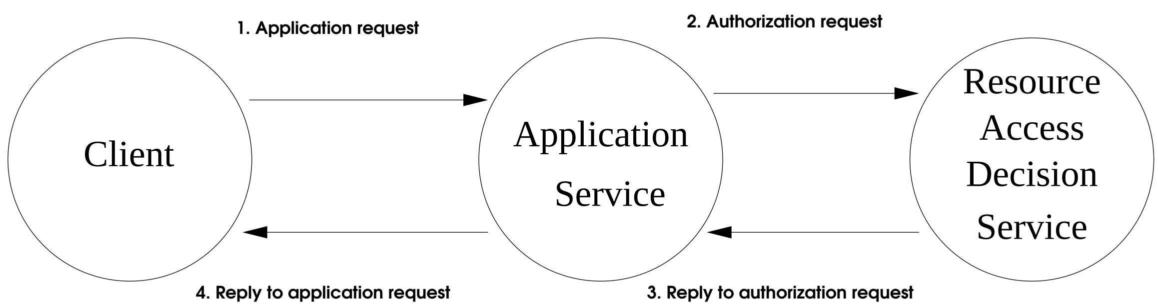

Figure 2. Interactions between client, application system, and access decision service.

In this case, security attributes are attributes of the current user session. The interesting parameters passed by RAD client are the first two: resource name and access operation. They are described below.

The RAD facility introduces an abstraction called “protected resource name” or just “resource name.” The resource name is used to abstract the application-dependent syntax and semantics of entities under application-level access control. A resource name can be associated with any valuable asset of an application owner, which is accessed by a client on behalf of a subject using it, and access to which is to be controlled according to the owner's interests. For example, electronic patient medical and billing records in a hospital are usually its valuable assets. The hospital administration is interested in controlling access to the records due to various legal, financial and other reasons. Therefore, the hospital administration considers such records as protected resources. Moreover, different information in those records count as different resources. Examples of different resources can be records from different visits or episodes for one patient. At the same time, a resource name can be associated with less tangible assets, such as computer system resources, including CPU time, file descriptors, sockets, etc. RAD does not interpret the semantics of the resource name. RAD only uses the resource name to obtain additional security attributes and to identify a set of policies that govern access to the resource associated by an application system with the resource name.

The access operation abstracts the access semantics of resources associated with resource names. An application may manipulate patient records on behalf of different care-givers, or may provide different hierarchies of menus to different technicians in the hospital. In either case, it is up to the application system developers and the enterprise security administrators to agree on semantics of the operation names used for each access. RAD does not interpret the semantics of access operations.

Before an application requests an instance of RAD for an authorization decision, it is supposed to identify the resource name and the access operation name associated with servicing the client request. There is no particular algorithm specified within RAD for performing such an association. For every application, or at least for every application domain, the way of associating protected entities with abstract resource names may be different. This provides the generality necessary for the RAD facility to be applicable to most of application domains.

5.2 Resource Access Decision Facility Design

The RAD facility is composed of the following elements:

- An Access Decider (AD) which receives requests for authorization decisions from RAD clients.
- Zero or more Policy Evaluators which provide evaluation decisions for those policies that govern access to the given resource. If a Policy Evaluator does not have any policy associated with the given resource name, the evaluator returns the result “don't know.”
- A Policy Evaluator Locator which provides references to potentially more than one Policy Evaluator.
- A Dynamic Attribute Service which provides dynamic attributes of the subject in the context of the intended access operation on the given resource associated with the specified resource name.
- A Decision Combinator which combines results of the evaluations made by Policy Evaluators into a final access decision by resolving evaluation conflicts and applying combination policies.

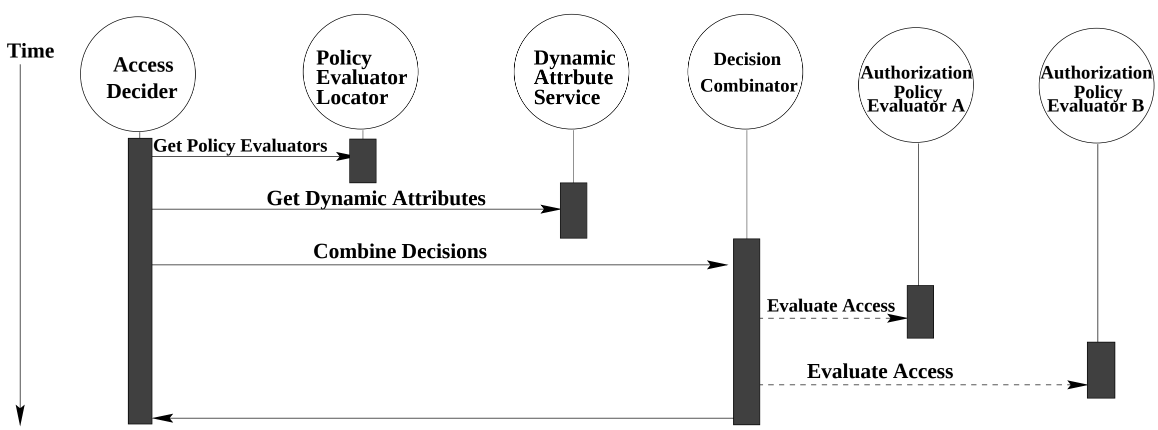

Figure 3. Authorization Service: Element Interaction

Figure 3 presents an interaction diagram of the RAD facility. Once the access decision service receives a request via the RAD interface, it makes an authorization decision according to the Algorithm 1. As illustrated in the figure and its description, the RAD interface performs the role of a Facade[11] to the service, i.e., it hides the complexity of RAD system from RAD client by presenting a higher-level easy-to-use interface.

Algorithm 1

1. Obtain the references to those Policy Evaluators that are associated with the resource name in question,
2. Obtain the dynamic attributes of the principal in the context of the resource name and the intended access operation on the resource,
3. Obtain the results from zero or more Policy Evaluators, and
4. Combine these decision results from step 3 into a final authorization decision.

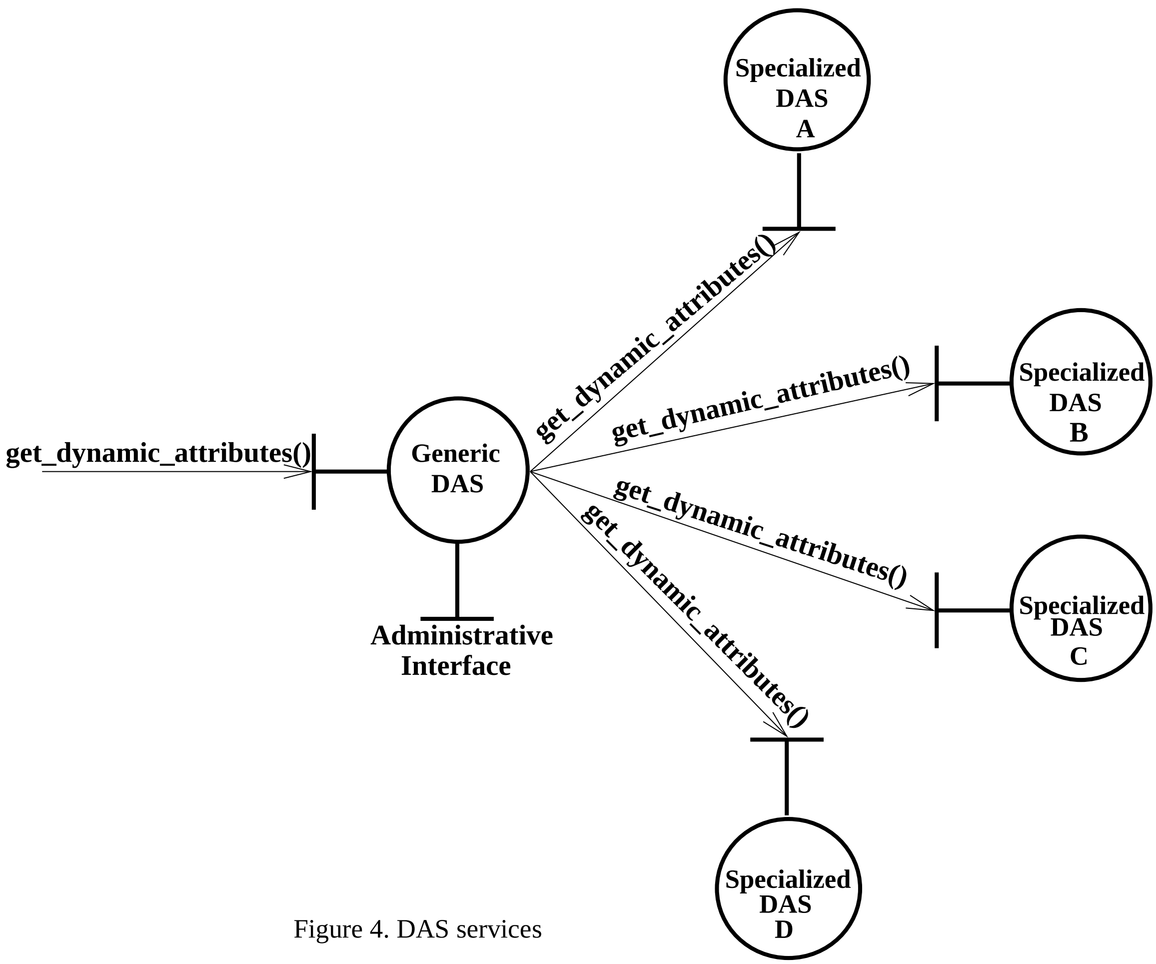

Figure 4. DAS services

One of the significant points of the RAD facility is the handling of factors specific to the application domain in the manner neutral to their semantics. All such factors are handled as dynamic[3] attributes. They are obtained from the enterprise environment via specialized dynamic attribute services. An access decision service does not interact with such services directly. It delegates the generic dynamic attribute service to collect all dynamic attributes from specialized dynamic attribute services. The semantics of a particular application domain (e.g., a patient/care-giver relationship) can be expressed in the form of dynamic attributes. This allows for the utilization of existing authorization mechanisms such as the traditional access matrix[12].

The objective of the dynamic attribute service (DAS) is to proxy several specialized DASs. We apply Proxy and Chain of Responsibility design patterns[11] for achieving this. Even though most of the work is done by specialized DASs, generic DAS represents them to the AD (Proxy pattern). RAD allows more than one specialized DAS object to resolve the dynamic attributes of the principal. This is possible because generic DAS decouples the AD and the specialized DASs. It provides the capability to issue a request to obtain the dynamic attributes without specifying the receiver of the request explicitly (i.e., the Chain of Responsibility pattern). The set of specialized DAS objects that can handle a request are specified dynamically via registering them with the generic DAS using an administrative interface.

[3] As opposed to the regular security attributes of the subject, which are called "static attributes" in the RAD specification.

Figure 4 illustrates the design of DAS services. Dynamic attributes are those attributes that express properties of a principal but are not administrated by security administrators. Dynamic attributes are so called because their values usually change more frequently than traditional user privilege attributes. Traditional “static” security attributes are used for describing relatively fixed properties of users. The values of static attributes are typically set by security administrators and are obtained by an application in an environment specific manner, e.g., from a principal's credentials in the case of an OMG Common Object Request Broker Architecture (CORBA) environment. While the use of a dynamic attribute in an access decision is determined by a security administrator, the values of dynamic attributes are usually set as part of normal business processes. In other words, dynamic attribute values are usually part of information content, not separately maintained security metadata whose values are set by a security administrator. Consequently, dynamic attribute values must be obtained at the time an access decision is required. This is in contrast to traditional “static” security attributes whose values are usually obtained when a session is established. The values of dynamic attributes may change during a session as a result of normal business processing.

Consider the following example of a dynamic security attribute. John Smith, a physician, attends patient Jane Doe. The physician has an attribute specifying such a relationship when a principal with access_id=johnsmith (speaking for John Smith) is accessing resources associated with medical records of patient Jane Doe. This relationship attribute is an example of a dynamic attribute in our model. It has the value “attending_physician” returned by a generic DAS only when John Smith accesses Jane Doe's records. The generic DAS obtains the value of this relationship attribute by consulting a specialized DAS, which computes the value of relationship attribute (probably by looking at the corresponding fields of Jane Doe's patient record which contains a list of Jane Doe's attending physicians). When John Smith is accessing resources not associated with any patient, this dynamic attribute of type relationship is not returned by the corresponding specialized DAS and consequently, it is not returned by generic DAS.

Another significant design element of the RAD facility is the encapsulation of authorization policies and their evaluators into separate entities. Such encapsulation is accomplished by means of the Policy Evaluators. Policy Evaluators can be considered either as distinct authorities, each representing a different set of authorization policies, or they can be considered as policy evaluation engines each supporting a particular policy language. Such a design insulates representation and interpretation of policies from the access decision service. It also allows adding and removing Policy Evaluators dynamically. By encapsulating the evaluation of those policies in Policy Evaluators, the design supports the implementation of arbitrary authorization policies.

Separation of concerns among various stakeholders involved in the authorization process (application developers, enterprise security administrators, access decision service developers) enables control of different factors in the authorization process by the appropriate parties:

- Application developers decide which functions of their application map into what access operations.
- User administrators control which users are assigned what static security attributes (e.g., roles).
- Implementers of the authorization services and other third party vendors control quality, performance, and other properties of the authorization service implementation.
- Workflow processes indirectly control which dynamic attributes are assigned to what users in the context of which resources.
- Security administrators administrate which access control policies govern what access to which named resources.

The generality of the RAD facility allows use of the authorization service in any application domain.

6 Example: Including Relationships within RBAC Access Decisions

As described in section 4, using an RBAC model to express relationships may be inefficient and non-intuitive. In addition, relationship information may already be kept as part of the information content associated with the business processes. Repeating such information within the RBAC mechanism would be redundant and error-prone. In this section, the use of RAD in conjunction with an RBAC Policy Evaluator and a Relationship Policy Evaluator is illustrated.

Consider a healthcare application which implements patient record access. There are two operations on patient records: *read* and *append*. Associated with the patient record repository is an access policy which includes the requirements: attending physicians may read and append information to the records of their patients; and hospital physicians, i.e., employed by the hospital, may read the records of hospital patients. Note that while these policy elements are realistic for the purposes of this example, an actual patient record access policy would be much more detailed and complex, e.g., attending physicians may only append to their sections of the patient record. Assume that the healthcare application is implemented within an information system environment where RBAC is the access control mechanism and the hospital maintains a database of patient records where the record for each patient lists those physicians who are currently treating the patient, i.e., the patient's attending physicians.

Applications that access the patient record database use the interfaces of the RAD facility to obtain an access decision based on the hospital's access policy. The RAD implementation references two Policy Evaluators (i.e., get_policy_evaluators() returns references to two Policy Evaluators: an RBAC Policy Evaluator that determines the user's roles, and a Relationship Policy Evaluator that determines authorization decisions based on relationships between the user and the individual patient who is the subject of the record). See figure 3, section 5.2.

The RBAC evaluator makes use of the static attribute "active_role_set." The Relationship evaluator makes use of the dynamic attribute "user/patient_relationships." The value of the static attribute active_role_set specifies the basic roles of a user, such as, physician, nurse, and registrar. In this example, the value of active_role_set is obtained from the security metadata in the user's credentials. The value of the dynamic attribute user/patient_relationships specifies the relationship between the user accessing the patient record and the patient who is the subject of the patient record being accessed,
e.g., "attending_nurse," "attending_physician," "consulting_physician." In this example, the value of the user/patient_relationships dynamic attribute is obtained by the Dynamic Attribute Service by accessing the content of the patient record which contains a list of attending physicians.

We now show the interaction between the application and the RAD implementation, and the internal processing steps of RAD for a specific invocation of the RAD interface. Figures 3 and 4 illustrate the sequence of steps within the RAD implementation. Assume that Doctor Smith is a staff doctor at the hospital, a hospital assistant administrator, and attending physician to Jane Doe. Doctor Smith has just examined Jane Doe and needs to enter clinical information into Jane Doe's patient record. Doctor Smith uses a browsing application to accomplish this.

The application program obtains from Doctor Smith the name of the resource to be read. It then obtains the static security attributes from Doctor Smith's credentials which includes the static attribute active_role_set whose value is "{physician}", i.e., "physician" is contained in Doctor Smith's active role set for this session. Doctor Smith's active role set does not contain the value "assistant_administrator" because although Doctor Smith is both a staff physician and an assistant administrator, hospital policy imposes a separation of duties requirement for that role pair. The roles "physician" and "assistant_administrator" may not both be active simultaneously in a session.

The application invokes access_allowed() (the principle interface from an application to the RAD) with the arguments patient_record_name, the operation "append," and the static attribute list. This procedure returns an indication of whether the physician is able to append to the requested patient record resource. If the physician has *append* access to the resource, the application appends the information and displays the updated resource for the physician.

The RAD now invokes get_policy_decision_evaluators() with the patient_record_name which returns:

- policy_evaluator_list that contains two items: the RBAC evaluator and Relationship evaluator names and references to their procedure calls.
- A decision_combinator.

The RAD then obtains dynamic attributes by invoking get_dynamic_attributes() with the static attribute_list provided by the application, patient_record_name, and the operation "append." This procedure returns a combined list of static and dynamic attributes consisting of the static attribute active_role_set whose value is "{physician}," and the dynamic attribute user/patient_relationships whose value is "{attending_physician}." The RAD then invokes combine_decisions() with patient_record_name, the operation "append," the combined list of static and dynamic attributes, and the policy_evaluator_list containing the RBAC evaluator and the Relationship evaluator.

Within combine_decisions():

1. The RBAC evaluator is invoked returning an access allowed indication since Doctor Smith has the static attribute active_role_set which includes the value "physician." To append to a patient record, a user must have a role which is permitted that operation.
2. The Relationship evaluator is invoked returning an access allowed indication since Doctor Smith has the dynamic attribute user/patient_relationships which contains the value "attending_physician." An attending physician can append information to a patient record.

Having invoked all evaluators in policy_evaluator_list, combine_decisions() returns an access allowed indication to the RAD since the rules for combining a decision from the RBAC evaluator and the Relationship evaluator require that both evaluators return an access allowed indication. In order to append to a patient record, the user must be active in the proper role and the user must have the proper relationship to the subject of the patient record.

Finally, the RAD returns an access allowed indication to the application. Since applications enforce the access decision returned by the RAD implementation, Doctor Smith's application appends clinical information to Jane Doe's patient record.

Now, assume that Doctor Jones is a hospital staff physician but is not an attending physician to Jane Doe. This being the case, Doctor Jones would have the static attribute active_role_set with a value of "{physician}" but the value "attending_physician" would not be contained in the dynamic attribute user/patient_relationships. As a result, if Doctor Jones attempted to append information to Jane Doe's patient record, the same sequence of events as described above would occur up to the point when the Relationship evaluator returns. In this case, since Doctor Jones does not have the value "attending_physician" in the dynamic attribute user/patient_relationships, the Relationship evaluator returns an access denied indication for the append operation. Consequently, combine_decisions() returns an access denied indication which results in RAD returning an access denied indication to Doctor Jones' application. As a result, the application refuses the request for the append operation from Doctor Jones.

7 RAD vs. Other Approaches

There have been several suggestions for expanding the RBAC model to include other factors beyond roles in a role based access decision[10][13][14][15]. While these approaches are effective, they can add elements to the basic RBAC model which may not have any direct relationship to the role metaphor. In general, what does a relationship between entities, or an attribute of an individual, e.g., age, or the time of day, or the location of the user requesting access necessarily have to do with an individual's role? These other factors most often have an importance in their own right.

User attributes and relationships are usually stored in an information system as part of normal business processing, not for the purpose of access control. Requiring such information to also be stored as security metadata creates redundant information managed by different administrators with different perspectives as to its use. In general, it is counter intuitive, both metaphorically and from a design perspective, to subsume such information within a role model. In many cases, the rationale for including other factors within a role model equally applies to their inclusion within other access control models. By using RAD to include other factors in a role based access decision, the role model retains its essential metaphor.

Another approach which is commonly used to include relationships and other factors in role based access decisions is to locate such access decision logic within application code. From a policy management point of view, this is very undesirable. Any time a policy changes, application code must be changed.

RAD enables RBAC to be the focus of access decisions when confronted with real world application implementation considerations. Implementing enterprise-wide applications usually requires implementing new applications within the context of several existing information systems. Each of these systems may have its own access control mechanisms administered by different parts of the enterprise. Because the role metaphor is the way in which enterprises are usually viewed by administrators, RBAC is the natural focal point for integrating access control among different access control mechanisms. RAD enables such integration while providing the capability for including information in the access decision which may not be a security attribute of any of the existing access control mechanisms. New evaluators and dynamic attributes can be added which allow any information, in particular relationship information, to be included in an access decision made at the enterprise level.

However, RAD may not be useful in all situations. The are many situations where there is no need for dynamic attributes, i.e., all factors used in an access decision are security metadata, and combining access decisions from different parts of an organization. The enterprise access policy may be straightforward and centralized. Furthermore, the ability to integrate access decisions made in different parts of an enterprise and the capability of including factors in access decision where the values of those factors are naturally very dynamic can result in unacceptable performance. The solution to the performance problem may not only require greater engineering effort applied to the enterprise system, but may also necessitate the development of less complex access control policies where possible.

8 Summary

The RBAC model and mechanism have proven to be useful and effective. Nevertheless, there are many common examples where access decisions must include other factors, in particular, relationships between entities, such as, the user, the object, the subject of the information contained within the object. This paper described the concept of relationships for use in access control, and showed how relationships can be supported using role based access control in conjunction with the OMG's RAD facility.

RAD allows dynamically changing relationships to be included in role based access decisions by providing the capability for:

- The separation of access control logic from application logic.
- The representation of relationship information as dynamic attributes whose values are set as a result of normal processing by applications.

- The combination of access decisions derived from a relationship access control model separate from the RBAC model.

This separation of access control models allows each model to retain its essential metaphor. Moreover, this separation often reflects real world organizations where an access policy is determined by the combination of policies administered by different divisions within an organization. Each division may have policies based on different metaphors.

References

[1] K. Beznosov, Requirements for Access Control: US Healthcare Domain. In Third ACM Workshop on Role-Based Access Control, October 1998.

[2] N.Adam, V.Atluri, E.Bertino and E.Ferrari, A Content-based Authorization Model for Digital Libraries, TR 98-104, CIMIC and MSIS Department, Rutgers University, 1998.

[3] Resource Access Decision (RAD), Object Management Group Healthcare Domain Task Force, Revised Submission, OMG TC Document corbamed/99-04-04, April 26, 1999.

[4] D. Ferraiolo, J. Barkley, and R. Kuhn. A Role Based Access Control Model and Reference Implementation within a Corporate Intranet. ACM Transactions on Information Systems Security, Volume 1, Number 2, February 1999.

[5] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role Based Access Control Models. IEEE Computer, 29(2), February 1996.

[6] K. North. Web Databases: Fun with Guests or Risky Business? Web Techniques, March 1999.

[7] R. A. Elmasri, S. B. Navathe, Fundamentals of Database Systems, Benjamin-Cummings Publishing Company, 1993.

[8] Eriksson, A., Penker M., UML Toolkit. John Wiley & Sons. 1998

[9] State of Florida Statutes. Hospital Licensing and Regulation, Chapter 395. 1998

[10] E.C. Lupu, M.S. Sloman, A Policy Based Role Object Model, First International Enterprise Distributed Object Computing Workshop, EDOC'97, Queensland, Australia, October 1997.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[12] Butler Lampson. Protection. In 5th Princeton Symposium on Information Science and Systems, pages 437-443, 1971.

[13] F. Chen, R. Sandhu, Constraints for Role-Based Access Control, First ACM Workshop on Role-Based Access Control, Gaithersburg MD, November 1995.

[14] L. Giuri, P. Iglio, Role Templates for Content-Based Access Control, Second ACM Workshop on Role-Based Access Control, Fairfax Virginia, November 1997.

[15] J. Barkley, Implementing Role-Based Access Control Using Object Technology, First ACM Workshop on Role-Based Access Control, Gaithersburg MD, November 1995.