A Model of OASIS Role-Based Access Control and Its Support for Active Security

JEAN BACON, KEN MOODY, and WALT YAO
University of Cambridge

OASIS is a role-based access control architecture for achieving secure interoperation of services in an open, distributed environment. The aim of OASIS is to allow autonomous management domains to specify their own access control policies and to interoperate subject to service level agreements (SLAs). Services define roles and implement formally specified policy to control role activation and service use; users must present the required credentials, in an appropriate context, in order to activate a role or invoke a service. All privileges are derived from roles, which are activated for the duration of a session only. In addition, a role is deactivated immediately if any of the conditions of the membership rule associated with its activation becomes false. These conditions can test the context, thus ensuring active monitoring of security.

To support the management of privileges, OASIS introduces *appointment*. Users in certain roles are authorized to issue other users with *appointment certificates*, which may be a prerequisite for activating one or more roles. The conditions for activating a role at a service may include appointment certificates as well as prerequisite roles and constraints on the context. An appointment certificate does not therefore convey privileges directly but can be used as a credential for role activation. The lifetime of appointment certificates is not restricted to the issuing session, so they can be used as long-lived credentials to represent academic and professional qualification, or membership of an organization.

Role-based access control (RBAC), in associating privileges with roles, provides a means of expressing access control that is scalable to large numbers of principals. However, pure RBAC associates privileges only with roles, whereas applications often require more fine-grained access control. Parametrized roles extend the functionality to meet this need.

We motivate our approach and formalise OASIS. We first present the overall architecture through a basic model, followed by an extended model that includes parametrization.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.2.1 [**Software Engineering**]: Requirements/Specifications; D.4.6 [**Operating Systems**]: Security and Protection—*access controls*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*

General Terms: Design, Security, Theory, Management

Additional Key Words and Phrases: Certificates, distributed systems, OASIS, policy, role-based access control, RBAC, service-level agreements

This article extends Yao et al. [2001], presented at SACMAT 2001.
Authors' address: Computer Laboratory, William Gates Building, JJ Thomson Avenue, University of Cambridge, Cambridge CB3 0FD, United Kingdom; email: Ken.Moody@cl.cam.ac.uk.

1. INTRODUCTION

Role-based access control (RBAC), in associating privileges with roles, provides a means of expressing access control that is scalable to large numbers of principals. The detailed management of large numbers of access control lists, as people change their employment or function, is avoided. RBAC is also likely to correspond well with policy expressed in legislation. We have worked on RBAC in the context of the design and deployment of large-scale, widely distributed applications. In such applications, policy is set by a loose federation of administrative domains, each of which has a degree of autonomy over the services that it manages. For example, a national Electronic Health Record (EHR) service may have a central EHR management domain, thousands of client domains (such as hospitals, clinics, research institutes and primary care practices) and millions of patients. Decentralized RBAC administered at the domain level is a realistic approach for such applications, in that it is not necessary to register all doctors nationally; a hospital domain may define a role *doctor* for its employees.

Large-scale, widely distributed applications are likely to contain independently developed and legacy software. Some policies are established across the application as a whole, whereas others are local to a particular administrative domain. Policies may restrict the use of services to suitably qualified principals and may specify constraints which must hold at the time a service is invoked. In our role-based, Open Architecture for Securely Interworking Services (OASIS) we assume that roles may be named and policies established at the level of individual services, so recognizing the heterogeneity that arises in such applications. We handle this diversity through service-level agreements (SLAs) established within and between administrative domains. The SLAs are implemented in the policies for role activation and service invocation, and, for each service, embody the requirement for credentials issued by other services. Incremental deployment is essential for large-scale, widely distributed systems; it is not feasible to install the access control system of a large-scale application atomically with a "big bang." Structuring an application into autonomous domains of independently developed services helps to make incremental deployment possible.

Pure RBAC associates privileges only with roles, whereas applications often require more fine-grained access control in order to handle relationships such as file ownership. Parametrized roles extend the functionality of RBAC to meet this need. The evaluation of role activation conditions can check parameter values and the relationship between them, so that, in particular, policy can express exceptions to the default access controls. For example, in the healthcare domain, a patient is empowered by law to specify individual exclusions such as "my uncle Fred Smith [who is a doctor] may not read my health record" which may be an exception to the default role-based policy. Parametrization is essential in order to make policy scalable and stable. *Role membership certificates* can include a number of typed parameters; exclusions can be checked against a database which forms part of the context during role activation or service invocation. Other constraints on context may relate to environmental constraints, such as

the time and place of the role activation or service invocation, as required by local or application-wide policy.

The design of OASIS was motivated by the need to satisfy access control requirements that are context-sensitive in large-scale systems, as in a national EHR service. Active management is needed because the context established at the start of a session may change. Suppose a doctor reports that a smartcard used for authentication into medical roles has been stolen; any currently active roles must be the result of impersonation. The card must be invalidated and the news propagated to all the services concerned, so that the roles can be deactivated and their associated privileges cancelled. The membership rule of a role indicates which of the role activation conditions must remain true while the role is active. A role is deactivated immediately if any of the conditions of the membership rule associated with its activation become false. This is facilitated by session-based role activation and implemented by building OASIS above an active, event-based, middleware platform [Bacon et al. 2000; Bacon and Moody 2002]. Event channels are set up between services, thus allowing all violations of membership conditions to be notified immediately.

Many RBAC schemes have used privilege delegation and role hierarchies, see Section 7. The essential requirement is that an authorized user should be able to issue credentials to other users in order to authorize them to perform certain actions. OASIS introduces the notion of *appointment*, whereby being active in certain roles authorizes the user to issue appointment certificates to other users. In order to transfer privileges by means of appointment certificates two separate privileges are needed; first, the right to enter an *appointer* role; second, the right to define security policy by establishing the current sets of *role activation* and *authorization* rules for a given service. This latter privilege is generic, and applies to any service protected by OASIS. It is natural to associate it with a *security officer* role, in the terminology of Sandhu et al. [1996, 1999]. Appointment certificates may be issued in many different circumstances, see Section 3, and their use is specific to each application. An example is that a hospital administrator need not be medically qualified yet may issue a credential which indicates that a user is employed as a doctor. This is a case of an *administrative role*, using the same terminology. A different example is that in an emergency situation a doctor may issue an appointment certificate to enable a junior colleague to activate a more senior role, which corresponds to *role delegation*. The lifetime of appointment certificates is independent of the issuing session, and they can therefore serve as long-lived credentials to represent academic and professional qualification or membership of an organisation. The role activation conditions of a service may include appointment certificates in addition to prerequisite roles and environmental constraints. An appointment certificate does not convey privileges directly but it is used as a credential for role activation. Appointment extends and abstracts role delegation.

Substantial work on RBAC as an effective means of replacing traditional discretionary and mandatory access control has led to the development of several models [Sandhu et al. 1996; Ferraiolo et al. 1999; Nyanchama and Osborn 1995]. Earlier papers about OASIS have defined the architecture and discussed large-scale, distributed system engineering issues. We have been developing a

practical implementation of OASIS for more than two years, during which a precise semantics, and therefore a formal model, became essential. A model is important for two reasons: first, it provides a sound foundation for reasoning about policy; and second, it acts as a reference framework to guide the implementer. In this article, we present a formal model for OASIS.

The remainder of this article is organized as follows. Section 2 introduces the OASIS model informally and relates it to the literature. Section 3 discusses *appointment*, the OASIS mechanism that enables the persistent allocation of privileges. Section 4 provides a formal description of the basic structure of the model in propositional logic, as presented in Yao et al. [2001]. Section 5 uses many-sorted first-order logic to extend this treatment with the syntax and semantics of parametrization. Section 6 develops scenarios which demonstrate applications of the model. Section 7 compares our approach to related work on active security. Section 8 briefly discusses the state of the OASIS implementation and our plans for evaluating the OASIS system in a practical application. Section 9 concludes the paper and points the way to future research that the model makes possible.

2. OVERVIEW OF THE OASIS MODEL

Central to the OASIS model is the idea of credential-based role activation. The credentials that a user possesses, together with side conditions that depend on the state of the environment, will authorize him or her to activate a number of roles. At any given time, a user will activate a subset of these potential roles in order to carry out some specific task, thus embodying the principle of least privilege in an organisation [Saltzer and Schroeder 1975]. The ability to activate and deactivate roles is vital to the support of active security [Thomas and Sandhu 1997], where the context is taken into consideration when an access is requested. The concept of role activation in OASIS is similar to the concept of session in Sandhu et al. [1996]. Activation of any role in OASIS is explicitly controlled by a *role activation rule*, and this rule may require that specified preconditions continue to hold while the role remains active, the *role membership rule*. Context-aware policy is therefore guaranteed to be satisfied throughout the period of activation of any role. In addition, the *authorization rule* a service enforces on an access request may require environmental constraints to be satisfied in addition to membership of a role, thus enforcing context-aware policy for service invocation.

A role activation rule specifies the conditions that a user must meet in order to activate a role. The intuition behind this is that roles are usually given to a person provided that he or she has met certain conditions, for example, being qualified as a physician, being employed by a company, being assigned to a task, being on shift, etc. We model these conditions in three categories, namely: prerequisite roles, appointments, and environmental constraints.

A prerequisite role in the condition for a target role means that a user must have already activated the prerequisite role before he or she can activate the target role. This is a session-based notion. The basis for the selection of prerequisite roles is competence and appropriateness [Sandhu et al. 1996].

Appointment occurs when a member of some role grants a credential that enables some user to activate one or more roles. The context may be an assignment of jobs or tasks. It may also be the passing of an examination, qualifying professionally, becoming employed, or joining an organization. An activation rule for a role depending on an appointment will require that the associated credential is presented when activating that role. Appointment can be used in many different ways, subsuming both role delegation and the use of administrative roles [Sandhu et al. 1996, 1999]. We describe the characteristics of appointment in Section 3.

Security policies in real life often involve constraints such as separation of duties. Several types of role constraint have been identified and discussed in the literature [Sandhu et al. 1996; Kuhn 1997; Simon and Zurko 1997; Gligor et al. 1998; Jaeger 1999; Nyanchama and Osborn 1999; Ahn and Sandhu 2000]. In our model, constraints may be associated with role activation rules, see Definition 4.4 and Section 5.2; in future work, we plan to specify role constraints at the organizational level, for example, "an account clerk cannot simultaneously be a billing clerk." We describe a possible implementation of certain types of role constraint in the discussion of negated prerequisite roles following Definition 4.6.

The use of roles allows access control policy to be specified in terms of the privileges of categories of users. This has two advantages: first, there is no need to change policy as staff come and go; second, details of individuals need only be taken into account during role activation. But in many applications it is insufficient to base access control decisions solely on roles and their assigned privileges. This is especially true when information such as time that depends on the context needs to be considered. Specific extensions have been proposed to the basic RBAC models in order to support workflow systems [Bertino et al. 1997; Kandala and Sandhu 2002] and team-based systems [Thomas 1997; Wang 1999]. More general ways of handling context are proposed in the content-based access control model of Giuri and Iglio [1997] and the generalized model of Covington et al. [2000].

In OASIS, we have extended the role model with parameters, based on first-order logic. Parameters may be included in the rules that cover both role activation and access to an object or service. Parameters may be bound to such items as the time of a role activation, the userid of a file owner, or an attribute of the object that is being accessed. The values that instantiate parameters are therefore context-dependent. Our model is similar to Giuri and Iglio's [1997] model based on *role templates*. A significant difference is that in OASIS it is possible to test context predicates during role activation as well as at the time of access. It may also be necessary to check the relationship between parameters; for example, a role for a primary care doctor might be parametrized with both the doctor's identifier and that of a patient. Suppose the role activation policy is that the patient must be registered with the doctor, and this is checked with an administration database at the time of role activation. If this relationship between parameters is also a membership condition, then any database update that invalidates the relationship must cause notification of the role-issuing service. We have extended the PostgreSQL database management system [Monjian 2000]

with active predicates for this purpose. In general, when a role is activated at a service an event channel is created in association with each membership condition. An event is triggered immediately any such condition becomes false, causing the role to be deactivated. Such a trigger may be generated when a timer expires or, as described above, when a database is updated to remove a user from membership of a group.

In Section 4, we present the details of a simplified model using propositional logic. This version omits parameters, but it covers most of the essential features of the architecture. In Section 5, we describe the full model, which is based on many-sorted first-order logic. This model includes extensions for handling parameter typing and parameter matching, and corresponds closely to our implementation. We adopt a formal approach using logic because adding parameters to privileges and roles adds a layer of complexity to the model. A logic-based approach helps to reduce errors in security policies by allowing static checks to be performed, for example, for completeness, consistency and reducibility. It also allows formal reasoning about security policies to discover potential errors or conflicts. The use of logic enables our model to be integrated with policies specified in pseudo-natural language. Preliminary, proof-of-concept work in this area can be found in Bacon et al. [2001a].

3. APPOINTMENT

In OASIS privileges are associated with membership of some role, which is active only for the duration of a session. In practice, it is essential to provide users with a means of establishing privileges that remains good from session to session. This is achieved through a level of indirection. The user is issued with a secure persistent capability [Gong 1989] that can serve as a prerequisite for role activation, thus allowing the user to obtain privileges repeatedly by activating a role. The capability does not itself convey any privileges, so limiting the damage if it should be compromised. A digital signature protects fields within the capability, which can safely be checked during role activation. We call this indirect technique for allocating privileges persistently *appointment*, and the corresponding capability an *appointment certificate*. After an appointment has taken place the appointment certificate is transferred to the user for whom it is intended, and it can then be used as a credential to activate one or more roles. As a side effect of creating the new appointment certificate a credential record (CR) is set up to enable its administration, and a reference to this CR is retained. The *validity rule* if any is stored with the CR, see Section 5.4.

Traditional delegation, within or external to a computer system, refers to the process whereby a principal transfers certain privileges to an agent to perform tasks on her behalf. The need to model the propagation of privileges in RBAC has led to work on role-based delegation [Barka and Sandhu 2000a, 2000b]. Role-based delegation is similar to traditional delegation, but instead of transferring privileges, roles are transferred. Because of the association of privileges with roles, privileges are effectively delegated through roles. While role-based delegation is a possible implementation of the temporary transfer of privileges and responsibilities [Crispo 1998], privileges are also granted to

principals in other ways, for example through the use of administrative roles. We discuss some of the ways in which appointment in OASIS extends delegation in Section 3.2.

3.1 OASIS Task Assignment and Qualification

In OASIS, we control the allocation of privileges by policies that allow roles to be activated. In particular, we can make possession of an appointment certificate a precondition of role activation. We consider two common bases for privilege allocation, namely *task assignment* and *qualification*, both of which we implement through the appointment mechanism.

Task Assignment. If a task is assigned to a principal by some authority, she will often require specific privileges to enable her to carry it out. In a role-based model, one could aggregate these privileges into a role and grant the right to activate this role to the principal.

As a motivating example, suppose that a doctor asks a nurse to place an order for drugs on her behalf (a privilege that a nurse does not usually have). To enable this type of task assignment, the security administrator could introduce a role *pharmacy-nurse* with the right to order drugs, and allow activation of *pharmacy-nurse* to a principal who is acting as a nurse and who has a valid recommendation from a doctor. In this case, the doctor who wishes to assign this task is an appointer, the "recommendation" is an appointment certificate, and the nurse who accepts this recommendation is an appointee.

Task assignment is in some ways similar to traditional delegation but there are important differences. In the above example, one might assume that the doctor herself has the privilege to order drugs. This is not always the case. For example, a screening nurse in the Accident and Emergency (A&E) department at a hospital assesses patients when they are admitted and assigns them to a particular doctor, basing the decision on both the needs of the patient and the current workloads of the medical team. The screening nurse need not have the privileges that a doctor possesses, but the role is *not* purely administrative. One effect of the assignment is to establish the treating doctor's right to see the patient's medical records. Task assignment is explicit about the *intention* of privilege propagation, whereas delegation focuses on the *process* of privilege propagation.

Qualification. Another common criterion for granting privileges is the possession of some valid qualification or credential. There are numerous examples in daily life. For example, in order to be employed as a doctor in the UK, one must be qualified by the British General Medical Council (GMC); in electronic commerce, a customer may need to produce a loyalty card in order to take advantage of special offers; at a university, in order to become fully registered, a student must prove that the tuition fee has been paid.

A fundamental difference from task assignment is that a *qualification* may not empower its holder with any specific rights, and its future uses may not be foreseen at the time of issue. The use of a qualification largely depends on the application. We expect a GMC qualification to convey privileges in a healthcare system in a hospital; it is unlikely to have value when buying goods

from an Internet shop, unless the GMC has come to an arrangement for the benefit of all doctors. Further, the degree of trust in a qualification is determined by the security policies of an application, which may be based on pragmatics, experience, or legal requirements. For example, in a healthcare system within the UK, a non-UK medical qualification may by default convey fewer privileges than its GMC counterpart.

A qualification in our model therefore refers to any credential that asserts certain facts. Academic and professional qualifications are examples; others include a receipt that establishes payment, a signed agreement or contract, a certificate of employment, a proof of identity or membership of a group.

Appointment is our abstraction for expressing policies relating to both task assignment and qualification. An appointment is a specific instance of either, represented by an appointment certificate. In task assignment, an appointment certificate can be seen as a *recommendation* or an *order*; in qualification, an appointment certificate represents the *qualification* itself.

The ability to express qualifications by appointment is an important feature, especially in a multidomain setting. Appointment certificates relating to task assignment embody the responsibilities associated with the appointment and their explicit existence allows the responsibilities to be recorded and audited.

3.2 Appointment, Role-Based Delegation and Administrative Roles

Both appointment and role-based delegation enable the propagation of privileges. In role-based delegation, this is achieved by delegating roles to some grantee; the unit is per role and not per privilege. There are many situations in which a principal wishes to give out only a subset of the privileges of a role. *Totality* is introduced in Barka and Sandhu [2000b] to identify whether the complete set of privileges assigned to a role is to be delegated; if the intention is to delegate only a subset, the authors refer to *partial delegation*. Partial delegation breaks the semantics of the strict RBAC model in order to delegate at the level of privileges. A partially delegated role shares an overloaded name with its delegating role, but it is in fact a new role that confers only a subset of the privileges. Any formal analysis must take this into account. A separate problem with the role delegation model is the restriction that a grantor can delegate only the privileges that she possesses. This is inadequate for situations in which the privileges relate to an object that is in some specific relationship to the grantee, such as the medical records of a patient undergoing treatment in the A&E department.

Role delegation is usually associated with task assignment, in cases where the principal assigning the task has the competence to carry it out. The appointment model avoids the problems of partial delegation. What an *appointer* grants is a credential that enables the activation of some role. The set of privileges associated with the new role may be a subset of those of the appointer role, as in the *pharmacy-nurse* example in Section 3.1. Establishing privileges that are specific to some object can be handled naturally through the use of parameters, which we discuss with many examples in Section 5. The appointment model and parametrization together make it easy

to define policies to allocate only the privileges that are intended, and at a fine granularity.

In many situations, the principal who is to allocate privileges does not herself possess them. For instance, a human resource manager in an organization is not allowed to touch the accounts, but she can authorize the appointment of an accounts clerk; in a hospital, a receptionist admits patients and assigns them to a doctor, but she does not have the privileges of a doctor. In the terminology of Sandhu et al. [1996, 1999], such functions are associated with administrative roles. Using appointment, since the privileges allocated will be derived from some role, the administrator can assign them by issuing an appointment certificate that is a prerequisite for activating the role in question.

Because of its support for fine-grained and controlled propagation of privileges, the appointment model embodies the principle of least privilege [Saltzer and Schroeder 1975]. The appointee may activate only those roles required to complete a task, subject to conditions restricting the context. Appointment in itself confers no privileges. Any privileges derive solely from roles activated on the basis of an appointment, and are limited to the current session.

Another issue in the delegation model is whether the grantee of a privilege should be allowed to delegate that privilege. Making delegation transitive leads to further complexity in the specification, with the introduction of *chained* delegation and revocation, and of the *permitted depth* of delegation. While chained delegation sometimes reflects real-life policies, these usually apply to the delegation of privileges at a fine grain. Transitive delegation of roles is unlikely to be appropriate in practice. In the appointment model, a principal in an appointer role grants an appointment certificate, so transitive delegation of roles does not arise; if transitive delegation of privileges is required, then policies can be defined to meet the need, choosing appointments and roles to suit the specific application. Transitive delegation of privileges in a role-based context is better expressed by appointment than by role delegation.

Role delegation can be viewed as a special case of appointment in which a user holding some role may appoint another user to activate that same role. For example, this mechanism could be part of the emergency procedure of a service when a role holder is called away or is taken ill.

3.3 Characteristics of Appointment

In Section 3.1, we presented a number of applications in which it is appropriate to use appointment. The possible uses of appointment differ in a number of ways. We characterize some of these differences in this section.

Type. Appointment captures task assignment and qualification using the same abstraction mechanism. It is often worth distinguishing the nature of an appointment. A *task assignment* appointment is usually transient, because a task will eventually finish (although it may require multiple sessions). A *qualification* appointment usually asserts some fact, so that it is more often persistent and long-lived.

Appointer. An *appointer* refers to a principal who initiates a process of appointment and issues an appointment certificate. Appointment can take place

only after the appointer has activated a role that includes the privilege of issuing appointment certificates. Let us refer to such a role as an *appointer* role. It is essential to constrain who may initiate an appointment, and the activation rules for *appointer* roles must specify the required policies. Three types of constraint on appointers should be considered: a set of *appointer* roles, a set of users, and a set of services.

In the case of *task assignment*, it is often easy to identify the potential appointers. The *appointer* role usually corresponds to the job entity responsible for task assignment. In such cases, anyone who is active in the appointer role can initiate the appointment process. However, there are some situations in taht the responsibility for a task lies with a specific user. It is easy in OASIS to constrain an appointer to a particular set of users. An example is an appointment that expresses the authorization to transfer money from an individual's account. Only the named account owner can initiate such an appointment. Discretionary access control in a filing system has similar semantics.

In the case of *qualification*, an appointment certificate is not issued for a single, explicit purpose. It is often the case that the appointer is an employee in a trusted position, who issues appointment certificates to assert group membership or academic credentials. For each type of qualification, there will be a service that can construct appointment certificates of the appropriate type. In order to do this, the trusted employee must activate an *appointer* role of that service.

Appointee. An *appointee* is a principal who receives an appointment certificate. It is essential to constrain who may use an appointment certificate in order to prevent fraud. This is achieved in OASIS by role activation rules. These rules specify prerequisite roles and environmental constraints which must be satisfied when an appointment certificate is used as a credential in order to activate a role. For example, parameter matching can ensure that the authenticated user in a session is the one to whom the appointment certificate was issued. In addition a *validity rule*, see Definition 5.8, can specify conditions which are checked whenever a particular appointment certificate is presented. These rules allow a broad range of policies to be expressed to control the use of appointment certificates.

For an appointment representing a qualification, it is possible to constrain its use only by naming the specific individual, since the contexts in which it will be presented are unknown at the time of issue. For task assignment, however, it is usually the case that an appointee must already be active in a specific role in order to make use of the appointment certificate. In OASIS, this can be enforced through the validity rule associated with the appointment certificate, see Section 5.4.

Revocation. When an appointment certificate is presented as a credential during role activation the issuing service validates it, checking the CR and the validity rule. The appointment certificate can be revoked by marking the CR as *no longer valid*. An appointment can be revoked in four ways: by its appointer only; by anyone active in the appointer role; by resignation; or by rule-based system revocation.

In the first case, an appointment can only be revoked by its appointer. This is common in real-life organizations; for example, the lead doctor in a care team might assign tasks to staff on that team by means of appointment. She then becomes responsible for monitoring their performance. Revoking the appointment of any member who performs badly is up to the lead doctor herself.

Dependence on a particular user to revoke may have undesirable consequences [Barka and Sandhu 2000a]; for example, if an appointer is ill or on leave, it may not be possible to take immediate action to limit damage. A more flexible solution is to allow anyone who can activate the *appointer* role to carry out the revocation. The principle is to reduce the risk by ensuring that it is always possible to revoke an appointment.

An appointee may also give up an appointment voluntarily by *resignation*. Resignation is only possible if the service issuing an appointment certificate makes explicit provision for it. Arguably resignation is redundant since an appointee can simply refrain from using an appointment certificate. Explicit resignation, on the other hand, allows the system to clean up any state associated with the appointment, and also prevents any future abuse of the appointment certificate.

The fourth possibility is *system-managed revocation*. In this case, an appointment is revoked automatically when certain conditions are met. There are many circumstances in which the revocation of an appointment can be handled by predefined rules. The conditions for system-managed revocation can be arbitrary predicates. Three common types of condition are based on time, task and session.

For *time-based revocation*, the appointer sets an expiry time when the appointment is made. The appointment certificate is revoked automatically at the expiry time. This is appropriate if the appointee has been employed on a fixed-term contract, or if the policy is to review long-lived credentials at regular intervals.

In applications such as workflow, an appointment may be conditional on the user's performance of specific *tasks*. The user only requires the privileges associated with a particular role while assigned to the relevant task; the system monitors progress, and once the task has been completed successfully, the appointment is automatically revoked. Continuing the A&E scenario, the lead doctor may appoint a member of staff to order a blood test and wish the appointment to be revoked once the order has been made. This approach, which requires substantial support from a task model, is suitable for a workflow environment.

The third type of system-managed revocation is based on *sessions*. The duration of an appointment can be restricted *either* to the session of the appointer *or* to the session of the appointee. In the former case, an appointment is valid so long as the appointer is still active in the appointer role. It will be revoked automatically when the appointer leaves the appointer role. An appointment can also be for the duration of the current session of the appointee. For example, a junior doctor may be appointed to stand in for a consultant who is called away to an emergency. When the junior's shift is over, the session and the appointment end.

Table I. Attributes of Appointment

Attribute	Option	Meaning
Type	Task-assignment	Appointment is related to some task.
	Qualification	Appointment represents some qualification or asserts some fact.
Appointer	Set of appointers	Only specific users in the appointer set may initiate this appointment.
	Appointer role	Anyone who can be active in the appointer role may initiate this appointment.
	Set of services	Appointment certificate is only valid if issued by any of the specified services.
Appointee	Specific users	The appointment certificate is only valid for certain specified users.
	Appointee role	Anyone who is active in a specified prerequisite role can use the appointment certificate.
	Unconstrained	There are no restrictions on the use of the appointment certificate.
Revocation	Appointer-only	Appointment revokable by the original appointer only.
	Appointer-role	Anyone who is actve in the appointer role may revoke.
	Resignation	The appointee may revoke this appointment voluntarily.
	System-managed	Time: revoked at expiry time.
		Task: revoked at the completion of the task.
		Appointer session: revoked at the end of the appointer session.
		Appointee session: revoked at the end of the appointee session.

A summary of these characteristics is shown in Table I. This list shows how a wide variety of practical security policies can be based on the use of appointments.

This range of policies can be implemented in OASIS. When an appointment certificate is used to activate a role it is first validated by the issuing service, which checks the CR and verifies the validity conditions, see Section 5.4. OASIS services are built on an active platform which deploys event-based middleware. Event channels may be built between services and an event may be signalled to an OASIS service immediately it occurs. Each active role has a membership rule, which indicates which of the activating conditions must remain true in order for the role to remain active. Should such a condition become false, this can be signalled immediately to the service. For example, if an appointment certificate is revoked by its appointer, the issuing service can notify any service at which that credential is a membership condition for some currently active role. Any such role is immediately deactivated.

4. BASIC MODEL

We present the formal model in the next two sections. In this section, we introduce the fundamental structure of the model based on propositional logic to formalize role activation conditions. It covers most of the ideas introduced in the previous sections, including appointment. We show in particular how to express the membership rules associated with active roles, and we explain how we enforce these rules using event channels.

OASIS role membership and appointment certificates include parameters. Role activation rules can match parameters to ensure, for example, that logged-in users can only invoke mutator methods on objects that they own. In Section 5, we describe the extensions required to handle parametrization. The extended model is based on many-sorted first-order predicate calculus, which allows the use of typed variables in expressions. Our models are not application specific. Instead, they are capable of expressing a variety of security policies across a range of applications.

4.1 Basic Constructs

The model is built on top of six basic sets, described as follows:

—$\mathcal{U}$: set of all user sessions[1]
—$\mathcal{S}$: set of all services
—$\mathcal{N}$: set of all role names
—$\mathcal{E}$: set of all environmental constraints
—$\mathcal{O}$: set of all objects
—$\mathcal{A}$: set of all access modes for objects (see Definition 4.2)

In addition to these sets, which are fundamental, two other sets are central to the interpretation of the basic model:

—$\mathcal{R}$: set of all roles
—$\mathcal{P}$: set of all privileges

A user is a human-being interacting with a computer system. Each user interaction takes place after an *authentication* which starts a *session*. An element in $\mathcal{U}$ can be any representation that uniquely identifies a user session within a system. In most situations the act of authentication identifies a particular user, and an audit trail can connect a user session $u \in \mathcal{U}$ to its human originator. In some circumstances there is a requirement for privacy, and it ought not to be possible to identify the human user behind a session. An example is described in Section 6.1.

The computer system is composed of a collection of services $\mathcal{S}$, which may be managed independently. A role is a named function that is associated with some service; a role is specific to a service and is defined below. Services confer privileges on their role members and may also recognize the roles of other services.

Definition 4.1. A role $r \in \mathcal{R}$ is a pair $(s, n) \in \mathcal{S} \times \mathcal{N}$, where $s \in \mathcal{S}$ is a service and $n \in \mathcal{N}$ is the name of a role defined by s.

The name of a role is unique within the scope of its defining service. When describing our model, we blur the distinction between roles and role names where this will not lead to confusion.

[1]Privileges are associated with active roles. Roles are deactivated automatically at the end of each authenticated session.

An environmental constraint $e \in \mathcal{E}$ is a proposition that is evaluated while making a security decision. Its value may depend on factors such as the time of day, the identity of the computer on which the current process is running or a condition such as group membership that requires access to a local database. In this article, we do not discuss the details of environmental constraints. We therefore consider each environmental constraint as an atomic proposition.

The conditions of some *role activation rule* must be satisfied when a role is activated. We may require in addition that some subset of these conditions, the *membership rule*, remains true throughout the session. If an environmental constraint e appears as a membership condition then its implementation must be *active*; when the role is activated each membership condition is evaluated, and in addition a trigger is set to notify the service should the condition become false. We discuss this requirement in more detail below.

A privilege is a right to perform some operation on a particular object. It is defined formally as follows:

Definition 4.2. A privilege $p \in \mathcal{P}$ is a pair $(o, a) \in \mathcal{O} \times \mathcal{A}$, where $o \in \mathcal{O}$ is an object and $a \in \mathcal{A}$ is an access mode for the object o.

The set of objects and their corresponding access modes are service dependent. For example, in relational database applications, objects may represent rows and their associated access modes include read- or update-attributes. In object-oriented systems, including distributed object systems, objects are represented naturally while access modes are the methods for each object. In general, we treat privileges as an abstract unit if the context permits. The underlying idea of RBAC is to associate privileges with roles, and roles with user sessions. These associations are described as relations in our model. Before describing these relations, we need to define the way in which roles are activated.

4.2 Appointment

We control the acquisition of privileges through role activation governed by rules. Roles can only be activated during a session, and being active in one role may be a precondition for activating another; an example is a log-in credential that ensures that the user has been authenticated. In order to activate certain roles a user must hold an appointment; the corresponding condition in a role activation rule is an *appointment certificate*.

Definition 4.3. An *appointment certificate* ω is an instance of an appointment. Each appointment certificate may be subject to a validity rule, comprising a set of prerequisite roles described by the function $\sigma : \Omega \to 2^{\mathcal{R}}$ and a set of environmental constraints described by the function $\eta : \Omega \to 2^{\mathcal{E}}$, where Ω is the set of all appointment certificates in the system.

An appointment certificate held by a user is valid only if the user is active in all of its prerequisite roles. This allows an appointer to ensure that an appointment certificate can only be used when the preconditions for activating all of those roles have been met. Its use is subject to the specified environmental constraints.

4.3 Role Activation

In order to activate a role during a session, a user must satisfy the conditions of some role activation rule. The formal definition is as follows:

Definition 4.4. A *role activation rule*, or activation rule for short, is defined as a sequent $(x_1, x_2, \ldots, x_n \vdash r)$, where x_j for $1 \leq j \leq n$ is an element in the universe $X = \mathcal{R} \cup \Omega \cup \mathcal{E}$, and $r \in \mathcal{R}$. We say that each x_j for $1 \leq j \leq n$ is an activating condition for the role r.

The sequent notation is conjunctive. In order to activate the role r through an activation rule $(x_1, x_2, \ldots, x_n \vdash r)$, a user must satisfy *all* conditions $x_1, x_2, \ldots, x_n$. *Satisfaction* interprets each element x_j within the current context to give a Boolean value, see Definition 4.6. There may be more than one activation rule associated with a particular role r.

An example of an activation rule is given below with $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$ and $\Omega = \{\omega_1, \omega_2\}$, where $\sigma(\omega_1) = (\{r_3\})$.

$$r_1,\ \omega_1 \vdash r_4.$$

According to this rule, a user who is active in role r_1 and holds the appointment certificate ω_1 can activate the role r_4, provided that the conditions for the appointment certificate to be valid are satisfied. In this case, the sole condition is that the user be active also in the prerequisite role r_3.

This definition of activation rule is essentially a restricted form of Boolean logic. Any Boolean expression without negation over the universe X can be translated into one or more activation rules by rewriting it into disjunctive normal form (DNF), and taking each implicant as an activation condition. For example, an expression in the same universe as the above example is shown below in Boolean logic syntax

$$(r_1 \vee r_2) \wedge \omega_1 \vdash r_4.$$

This is translated to DNF,

$$(r_1 \wedge \omega_1) \vee (r_2 \wedge \omega_1) \vdash r_4,$$

which can be written in sequent notation as shown below

$$r_1, \omega_1 \vdash r_4$$
$$r_2, \omega_1 \vdash r_4.$$

The set of all activation rules specified in a system is denoted by Γ. We summarise the symbols representing additional sets of objects in our model here.

—Ω: set of all appointment certificates
—Γ: set of all role activation rules
—Λ: set of all membership rules see Section 4.4

We now consider a special type of role activation rule, called initial.

Definition 4.5. A role activation rule $(x_1, x_2, \ldots, x_n \vdash r)$, in which for $1 \leq j \leq n$ the element $x_j \in \Omega \cup \mathcal{E}$, is initial. The role r is said to be an initial role.

Initial rules provide a means to allow users to start a session by acquiring initial roles. A particular case is that of rules with no antecedent conditions, $\vdash r$. The activation of such an initial role depends on system policies and typically requires a system-dependent mechanism, for example, biometric or challenge-response authentication. In general, activation of an initial role may require an appointment certificate (in this case having no prerequisite roles, see Definition 4.3) and be subject to environmental constraints. The set of all initial roles is denoted $\mathcal{IR} \subseteq \mathcal{R}$. We restrict explicit association between users and roles to initial roles. In order to activate any other role a user must satisfy the preconditions of some activation rule, including possession of one or more prerequisite roles. These preconditions can include appointments and environmental constraints as well as role membership.

Note that during a session a user accumulates privileges by activating a succession of roles. Starting from a set of initial roles, which become active following authentication, a number of roles may be entered according to specified rules. An acyclic directed graph structure is therefore established that exhibits the run-time dependency of each role on its preconditions. Superficially, the structure is similar to a static role hierarchy, but there are important differences. First, the dependency structure is dynamic; there may be several activation rules for a given role, but a particular activation depends on a single rule only, the *current activation rule* for that role. Second, any privileges acquired by entering a role in this way will usually not be shared with any prerequisite role; it is likely that the new role is more specific and has been activated on the basis of appointment, or perhaps after checking a database, see Section 5.3.

At present, an active role is deactivated implicitly at the end of a session, or if a *membership condition* for the current activation should become false, see Section 4.4. It would be easy to provide an interface for a session holder to deactivate a role explicitly, so giving better support to the principle of least privilege [Saltzer and Schroeder 1975].

The activation of a non-initial role requires a user to satisfy each of the conditions of some activation rule for that role. We define what is required for elements of each of the sets $\mathcal{R}$, Ω and $\mathcal{E}$ to satisfy a precondition for role activation.

Definition 4.6. The interpretation function for a role activation rule is a truth assignment with type, $I : X \to \{\mathbf{true}, \mathbf{false}\}$. An interpretation function, I, with respect to a user session $u \in \mathcal{U}$ is denoted as I_u, and is defined below:

$$I_u(x) = \begin{cases} \mathbf{true} & \text{if } x \in \mathcal{R} \text{ and } u \text{ is active in role } x, \\ & \text{if } x \in \Omega, u \text{ possesses the appointment certificate } x, \\ & \quad u \text{ is active in all the prerequisite roles } r \in \sigma(x), \\ & \quad \text{and all environmental constraints } e \in \eta(x) \; hold, \\ & \text{if } x \in \mathcal{E}, \text{ and the evaluation of } x \text{ yields } \mathbf{true}. \\ \mathbf{false} & \text{otherwise} \end{cases}$$

Note that the definition of activation rules does not include negation ($\neg$). We briefly consider the effect of allowing negation of each of the three types of role activation condition. First, environmental constraints $e \in \mathcal{E}$ are atomic by definition. Any discussion of negation must take place in the context of an

explicit environmental sublanguage, such as temporal expressions that test the time of day.

Second, appointment certificates $\omega \in \Omega$ record the fact of an appointment. It is only when the appointment certificate satisfies a precondition for activating a role that any privileges are bestowed on the user. Negation should be associated with the roles activated rather than with the appointment certificate itself. In any case, appointment certificates can be anonymous and therefore transferable from user to user, as might be the case when joining an organization. The advantage of such a scheme is that one credential record covers all the users of such an appointment certificate, which makes it possible to revoke the appointment of every member by a single action. It is not in general possible to tell whether a user has such an anonymous appointment.

Negating a role $r \in \mathcal{R}$ makes perfectly good sense. Indeed, allowing a negated role among the conditions for role activation has a natural interpretation under I_u, namely $I_u(\neg r) = \textbf{true}$ if u is NOT active in role r. This is a possible implementation of a separation of duties constraint. But if a user must not activate two roles simultaneously, then the activation rules for each role should indicate that this user must not be active in the other. A more appropriate way of specifying the requirement would be to declare an explicit separation of duties constraint. That is beyond the scope of this article.

Given the interpretation function, we can then define role activation formally.

Definition 4.7 (*Role Activation*). A role $r \in \mathcal{R}$ can be activated within a session $u \in \mathcal{U}$ by the activation rule $\gamma = (x_1, x_2, \ldots, x_n \vdash r) \in \Gamma$ provided that $I_u \models x_j$ for all $1 \leq j \leq n$, where I_u is the interpretation function for u at the time when the activation request is made. $\gamma \in \Gamma$ becomes the *current activation rule* for role r.

Note that the definition of the interpretation function implies that its evaluation with respect to a user session changes with the context. When a user requests an activation of a role, the interpretation function is immediately evaluated in the current context and the decision is made.

4.4 Membership Requirements and Role Deactivation

The opposite of role activation is role deactivation. Often continuing activation of a role will be valid only if some subset of the activation conditions continues to hold. These are called the membership conditions. The *membership rule* associated with a role activation rule specifies those conditions that must remain true in order for a user to remain active in that role.

When a role is activated at a service $s \in \mathcal{S}$ each of the conditions of the activation rule is verified. For roles associated with s itself, this is straightforward. Roles and appointment certificates of other services must be validated by the issuer. In the case that x_i is a membership condition s establishes an event channel on the trigger $\neg x_i$ so that the issuer can notify s should the condition become false. OASIS depends on asynchronous notification to support role deactivation, see, for example, Bacon et al. [2000].

We have not defined explicit sublanguages for environmental constraints in this paper. However, it is worth considering two examples. First, let us suppose

that a particular role may be held only between 16:00 and 18:00 hours on any day. We can include this requirement as part of the activation rules through a constraint in $\mathcal{E}$; at activation, we check the time of day, say 17:23, and set a timer exception for 18:00 hours. In this instance, the evaluation is independent of the user session u.

Second, suppose that a principal in session u requests a privilege that is restricted to members of group g. For this example, we require active database support. At activation, we check the database to ensure that the user responsible for the session is a member of the specified group. At the same time, we set a trigger for the negation of the condition. If the group manager updates the database to exclude the user, then the trigger fires and deactivation takes place. This example shows how constraints in $\mathcal{E}$ may be user specific. The first prototype implementation of OASIS included a simple associative tuple store with triggers.

Definition 4.8. The *membership rule* associated with the activation rule $(x_1, x_2, \ldots, x_n \vdash r) \in \Gamma$ for the role $r \in \mathcal{R}$ is the sequent $(x_1, x_2, \ldots, x_m \vdash r)$ for some $m \leq n$, where x_i for which $1 \leq i \leq m$ are the membership conditions.

Associated with any active role, there is a current activation rule, see Section 4.3. We refer to the associated membership rule as the *current membership rule*. An active role r shall be immediately deactivated if the current membership rule can no longer be satisfied. We denote the set of all membership rules in a system as Λ. The formal definition of role deactivation is given below.

Definition 4.9 (*Role Deactivation*). Suppose that a role $r \in \mathcal{R}$ has current activation rule $\gamma \in \Gamma$, and that $(x_1, x_2, \ldots, x_m \vdash r) \in \Lambda$ is the corresponding membership rule. Then r shall be deactivated as soon as the context changes so that $I_u \not\models x_i$ for some membership condition x_i, where I_u is the interpretation function for u.

A principal will be active in role r only while the current membership rule continues to be satisfied. Note that deactivation of a role r may trigger deactivation of some other role r' for which being active in r is a membership condition. This is referred to as *cascading deactivation*. Its implementation is discussed in Bacon et al. [2000] and Hayton et al. [1998], which describe the use of an event infrastructure.

Note that the membership rule associated with an active role r is specific to the rule under which r was activated. Consider as an example the rules obtained by translating the Boolean expression introduced after Definition 4.4:

$$(r_1 \vee r_2) \wedge \omega_1 \ \vdash \ r_4,$$

which specifies that a user who is active *either* in role r_1 *or* in role r_2 and who holds an appointment certificate ω_1 may activate role r_4.

The corresponding activation rules are as follows:

$$r_1, \omega_1 \ \vdash \ r_4$$
$$r_2, \omega_1 \ \vdash \ r_4.$$

In each case, the membership rule will include the relevant prerequisite role, so as to enforce cascading deactivation at the end of the session. If revocation of the appointment is to take immediate effect, then the appointment certificate must also be a membership condition.

4.5 Dealing with Privileges

We can now define the association of roles with privileges. This is expressed as a relation as follows.

—RP $\subseteq \mathcal{R} \times \mathcal{P}$, the role-privilege relation.

RP describes the role-privilege relationship. It is a many-to-many relation specified by the security administrators of an organisation to express security policies. We distinguish two sets of privileges for a role by the terms *direct* and *effective*. Our definitions are different from those given in Nyanchama and Osborn [1999], where direct and effective privileges are defined with role hierarchy in mind.

The *direct privilege set* of a role $r \in \mathcal{R}$ is the set of privileges assigned to r directly, that is, $\mathrm{DP}(r) = \{p \mid (r, p) \in \mathrm{RP}\}$. The *effective privilege set* of a role r is the set of privileges that continue to hold within a session while a user is active in role r. This includes the effective privileges of all roles specified as membership conditions when r was activated, as well as the prerequisite roles of any appointment certificates. Each of these roles must still be active, or r would have been subject to cascading deactivation. The effective privilege set is dynamic, and depends on the specific activation history. The following definition ascends the activation tree recursively.

Definition 4.10. Suppose a user holding a session $u \in \mathcal{U}$ is active in some role r whose current membership rule is $(x_1, x_2, \ldots, x_m \vdash r)$. The effective privilege set $\mathrm{EP}(r)$ of r is defined as follows:

$$\mathrm{DP}(r) \cup \bigcup_{1 \le i \le m} \begin{cases} \mathrm{EP}(x_i) & \text{if } x_i \in \mathcal{R} \text{ is a prerequisite role} \\ & \text{if } x_i \in \Omega \text{ is an appointment certificate, the union of} \\ \mathrm{EP}(\rho) & \text{for all prerequisite roles } \rho \in \sigma(x_i). \end{cases}$$

In some RBAC models, it is possible to compute the maximum privileges that a user may assume. OASIS defines security policies on a service-by-service basis for multiple management domains in a distributed world. For example, a nationwide system for electronic health records will comprise many interoperating domains such as hospitals, primary care practices, clinics, research institutes etc. Services within a given domain express their policy for role activation and service use. Membership of a role of one service may be required as a credential for entering another. Such dependencies are specified in service level agreements. It is likely that policy will be administered at domain level, and will derive from local and national administrative and legal sources, depending on the application. Service level agreements will also be made across domains. Appointments may be made at several administrative levels. Some appointment certificates will apply to many domains, for example those

representing academic and professional qualifications. Others will be dynamic and local, for example temporary substitution for a colleague who is called away while on duty.

Should it be required, it is possible to compute the maximum privileges that a user may obtain based on statically known appointments. This assumes that all constraints will be satisfied at the time roles are activated. In practice, dynamic environmental conditions may prevent some roles from being activated in any specific session. In addition, unforeseen appointments might be made dynamically within sessions.

4.6 Managing Appointment

Previously we introduced *appointment certificates* to represent appointments. Services that support appointment will define their own roles and policies to manage it, and will issue and validate the appointment certificates. At each appointment, an appointment certificate is returned to the appointer who subsequently transfers it to the appointee. The latter can then use the appointment certificate during role activation, either at the issuing service or at some other. A role activation rule may specify a number of prerequisite roles in addition to one or more appointment certificates. In this way, we can, for example, implement the two-signature, countersign approval system commonly found in business by requiring two appointment certificates. In addition, the appointer, when applying for the appointment certificate, may specify a set of prerequisite roles in which the appointee must be active and a set of environmental constraints, see Definition 4.3.

OASIS supports rapid and selective revocation, which is managed by invalidating the certificate issued on an appointment. When an appointment is made a credential record is created, which is checked whenever the appointment certificate is validated. Subsequently, the appointment certificate can be revoked by setting the credential record to show that it is no longer valid. If an appointment certificate is a current membership condition for an active role, then an event channel is established at the time of validation. The use of events to implement cascading revocation for RMCs is described in Bacon et al. [2000]. A revocation certificate is a capability that identifies the credential record for the corresponding appointment certificate. The use of revocation certificates may be restricted in the ways listed in Table I. This basic model is sufficient to support system-managed revocation.

5. EXTENDED MODEL

In the basic model described in Section 4, access control decisions are made on the basis of propositions evaluated in the current context. These propositions relate to roles and appointments, and the policy governing the acquisition of privileges is expressed in terms of them. Role activation rules can take account of the execution environment by evaluating propositions relating to factors such as the current time of day or an entry in an administration database. We extend this model to allow parametrization of roles, appointment certificates, privileges and environmental constraints. We accommodate these extensions

by defining role activation rules and membership rules in terms of predicates rather than propositions.

5.1 Basic Syntactic Constructs

The extended model shares many of the sets found in the basic model, namely $\mathcal{U}$, $\mathcal{S}$, $\mathcal{N}$, $\mathcal{O}$ and $\mathcal{A}$. In this section, these symbols refer to sets in the extended model rather those in the basic model.

We base the formalism on first-order logic, extended to include parameter typing. More precisely, our model is a simple form of many-sorted first-order logic. There are two main reasons for including types in the model. First, parameter typing aids intuition and so makes policies more readable. The second consideration is practical. We have an implementation of OASIS based on XML/Schema [Thompson et al. 2001] and Simple Object Access Protocol (SOAP) [Box et al. 2000], which provides access control enforcement for SOAP services. Access control decisions can be based on the actual parameter values of a SOAP request. The type system in this implementation is that of XML/Schema [Biron and Malhotra 2001]. Parameter typing is a great help in this case because it allows static checks to be performed.

Let $\mathfrak{T}$ be the set of all types. For a parameter v of type $t \in \mathfrak{T}$, we write v^t. We assume there is an unlimited supply of parameters, which may be denoted using subscripts, $v_1, v_2, \ldots$. The set of all parameters of type t is $\mathfrak{V}_t$, that is, $\mathfrak{V}_t = \bigcup_i v_i^t$, where $t \in \mathfrak{T}$. We let $\mathfrak{V}$ be the set of all possible parameters,

$$\mathfrak{V} = \bigcup_{t \in \mathfrak{T}} \mathfrak{V}_t.$$

Our vocabulary consists of three sets: a set of typed parameters $\mathfrak{V}$, a set of typed function symbols $\mathfrak{F}$, and a set of typed predicate symbols $\mathfrak{P}$. Function and predicate symbols, each of some arity $n \geq 0$, are typed by means of a *signature*.

Definition 5.1. We define the signature $\Sigma(x)$ of function and predicate symbols as follows:

— If $f \in \mathfrak{F}$ denotes an n-ary function $f(u_1^{t_1}, \ldots, u_n^{t_n})$ and f has type t, $\Sigma(f) = (t_1 \times \cdots \times t_n) \rightarrow t$.

— If $P \in \mathfrak{P}$ denotes an n-ary predicate $P(u_1^{t_1}, \ldots, u_n^{t_n})$, $\Sigma(P) = (t_1 \times \cdots \times t_n) \rightarrow$ *bool*.

A constant symbol c of type t is treated as a 0-place function with a signature $\Sigma(c) = () \rightarrow t$. Contrary to what might be thought, because of the nature of our model a constant symbol does not necessarily bind semantically to a constant value. All expressions are evaluated in a specific context and the value of a constant symbol may be context-sensitive, for example, *current_time*. We shall return to this when we explain the activation model in Section 5.6.

As in first-order logic the syntax of our model comprises terms and rules. Terms refer to individual entities of a system, whereas rules define associations between the various components of the model, for example, appointments, roles

and privileges. Rules are a set of specialized formulae in first-order logic which are given semantics that corresponds naturally to the OASIS implementation.

Parameters in our model have two modes, *in* and *out*. These mode names are motivated by the programming model. Predicates are evaluated in some context during the interpretation of a rule. The value of an in-parameter must already be set before predicate evaluation, whereas the value of an out-parameter is established as a side-effect during the process of evaluation itself, for example, by pattern matching, calculation or database lookup. In our notation, we denote the occurrence of variable x as an out-parameter by $x?$, and as an in-parameter simply by x, for example, $P(x, y?)$ features x as an in-parameter and y as an out-parameter in the predicate $P(v_1^{t_1}, v_2^{t_2})$. Note that the modes refer to the flow of information through individual predicates, *not* to the flow through the engine that evaluates rules. Parameter resolution is straightforward, since the semantics of rules is such that an efficient plan for each rule can be determined statically, see Section 5.6.2.

Before describing the syntax of predicates, we establish a many-sorted algebra of typed terms.

Definition 5.2. A term of type t is defined recursively as follows:

— If $v^t \in \mathfrak{V}_t$ is a variable, then an in-parameter v^t is a term of type t.
— If $u_1^{t_1}, \ldots, u_n^{t_n}$ are terms and an n-ary function $f \in \mathfrak{F}$ has signature $(t_1 \times \cdots \times t_n) \rightarrow t$, then $f(u_1^{t_1}, \ldots, u_n^{t_n})$ is a term of type t.
— In particular a constant symbol c of type t is a term of type t.

For the sake of clarity, we drop the type superscript in a parameter name in situations where types have no effect on the formalism. But it is crucial to remember that all parameters, functions and predicates are typed.

In our model, rules are interpreted in a particular context. Parametrized predicates correspond in the interpretation to roles, appointment certificates, privileges and environmental predicates. Predicate symbols are therefore partitioned into four sets, $\mathcal{R}$, Ω, $\mathcal{P}$ and $\mathcal{E}$. The first three types correspond to access control system entities. Out-parameters are set by pattern matching from specific instances of role membership and appointment certificates; during role activation fields in the role membership certificate (RMC) are set by evaluating terms that may include in-parameters. Environmental predicates are implemented by computational procedures such as database lookup that can be invoked when a rule is interpreted. Only environmental predicates may contain both in- and out-parameters. The corresponding predicate expressions share the same general syntax:

Definition 5.3. If $P \in (\mathcal{R} \cup \Omega \cup \mathcal{P} \cup \mathcal{E})$ is a predicate symbol of signature $(t_1 \times \cdots \times t_n) \rightarrow \mathit{bool}$ and $u_1, \ldots, u_n$ are expressions of types $t_1, \ldots, t_n$, then $P(u_1, \ldots, u_n)$ is an n-ary predicate expression.

When the parameter values are unimportant we may refer to a function or predicate expression simply by its symbol. The precise forms allowed for the parameter expressions $u_1, \ldots, u_n$ will depend on the context.

5.2 Role Activation Rules

The syntax for specifying role activation rules follows Definition 4.4 in the basic model almost exactly. We use predicate expressions in place of propositional variables and we introduce constraints on the way in which parameters can be used. A role activation rule has syntax:

$$(x_1, x_2, \ldots, x_n \vdash r),$$

where each x_j for $1 \leq j \leq n$ (an *antecedent*) is a predicate expression derived from some predicate symbol in the universe ($\mathcal{R} \cup \Omega \cup \mathcal{E}$), and r is the target role predicate expression. We now define additional syntactic constraints on the parametrizations.

Definition 5.4. The parameters that may appear in the predicates of role activation rules are restricted as follows:

— If an antecedent $x_j = \chi(u_1, \ldots, u_n)$ for some predicate symbol $\chi \in (\mathcal{R} \cup \Omega)$, then each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a constant c of type t_i.

— If an antecedent $x_j = \epsilon(u_1, \ldots, u_n)$ for some predicate symbol $\epsilon \in \mathcal{E}$, then each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a term of type t_i.

— The target role predicate expression $r = \rho(u_1, \ldots, u_n)$ where $\rho \in \mathcal{R}$ and each $u_i^{t_i}$ is a term of type t_i.

We may require that some subset of the activation conditions (the *membership conditions*) continue to be satisfied while the role remains active. The details are essentially the same as in the basic model, see Section 4.4, and in particular Definition 4.8. Each membership condition must have an active implementation.

We next define parameter binding for our model. Intuitively, the values taken by bound variables are determined precisely when a role activation rule is interpreted in a specific context.

Definition 5.5 (*Bound and Free Variables*). A variable u is bound in a rule $(x_1, x_2, \ldots, x_n \vdash r)$ if at least one occurrence of u in the rule is as an out-parameter. A variable u is free in a rule if it is not bound.

For instance, consider an activation rule $\rho(u?), \omega(73, v?), \epsilon(v, w) \vdash \tau(w)$, where $\rho \in \mathcal{R}$ is a prerequisite role, $\omega \in \Omega$ is an appointment certificate, $\epsilon \in \mathcal{E}$ is an environmental predicate and $\tau \in \mathcal{R}$ is the target role. The variables u and v are bound, while w is free because it does not occur as an out-parameter. If the mode of the parameter w in the environmental predicate ϵ were altered to *out*, then w would also be bound in the rule. Here 73 is an integer term, a 0-ary function whose evaluation is *not* context-sensitive!

CONSTRAINT 1. *There must be no free variables in an activation rule*

$$(x_1, x_2, \ldots, x_n \vdash r).$$

The reason behind this constraint will become clear when we explain the semantics of activation rules. By excluding free variables from the rule we

establish clearly defined activation semantics.[2] The semantics of parameter handling is similar to the unification process found in Prolog. Note that we can exploit the side effects of predicate evaluation to set parameters in the target RMC to context-sensitive values, thus supplying application-dependent attributes to the underlying security mechanism. We first explain the semantics informally with the aid of examples, later giving a formal description.

Example 1. Suppose we are modelling a hospital policy that says that *a user who is employed there as a doctor may acquire a doctor_on_duty role whenever she is on duty.* For simplicity, we do not represent the wards, clinics or departments to which the doctor is allocated in these first examples. (An extension with the doctor's current work location as an additional parameter is shown in activation rule 4 of Section 6.2.) We define the following predicates:

Name	Type	Parameters
local_user(*h_id*)	role	*h_id*: local user id
employed_medic(*h_id*)	appointment	*h_id*: local user id
on_duty(*h_id*)	environment	*h_id*: local user id
doctor_on_duty(*h_id*)	role	*h_id*: local user id

We make the following assumptions about the hospital security policy. First, all hospital employees are authenticated into the role *local_user*(*h_id*) when they log in. Second, an appointment certificate *employed_medic*(*h_id*) is issued whenever a qualified doctor is first employed. Finally, the condition *currently on duty* is checked by the environmental predicate *on_duty*(*h_id*), here implemented by consulting a database. We have extended the standard PostgreSQL implementation to act as an active predicate store, see Section 5.6.1. Were *on_duty*(*h_id*) to be a membership condition of *doctor_on_duty*, the role would be deactivated when *h_id* came off duty. For the purposes of this example, we could make the unrealistic assumption that the time a doctor is to come on and off shift is recorded statically in the database and enforced. In a more enlightened world, the doctor might wear an active badge and be tracked in a sensor-rich hospital environment. The database would be updated dynamically when she was detected entering the medical ward area (so on-shift) and leaving it (so off-shift).

The policy can be expressed as follows:

$$local\_user\,(h\_id\,?),\ employed\_medic\,(h\_id\,?),\ on\_duty\,(h\_id\,) \vdash\ doctor\_on\_duty\,(h\_id\,).$$

This rule says that a doctor may acquire the role *doctor_on_duty* when she is working. The values of *h_id* set by pattern matching while checking the preconditions *local_user*, *employed_medic* must be equal for this rule to succeed. This common value is passed to the environmental predicate *on_duty*, which checks that the doctor is on shift. Note that in the latter context *h_id* is regarded

[2]Some extra static checks may be needed to ensure that no cyclic dependencies can arise during parameter matching, see Section 5.6.2.

as a term. Finally, this value is used to initialise the target RMC. This rule corresponds to the first-order logic formula:

$$\forall x\,(local\_user\,(x) \wedge employed\_medic\,(x) \wedge on\_duty\,(x) \rightarrow doctor\_on\_duty\,(x)).$$

Example 2. Continuing the example, suppose another policy states that *all patients in a ward are in the care of the doctor currently on duty there*. Again, for simplicity, we do not represent the ward as a parameter, but see Section 6.2. We extend the system to include the following predicates:

Name	Type	Parameters
$treating\_doctor(h\_id, pat\_nhs\_id)$	role	h_id: local user id, pat_nhs_id: patient id
$ward\_patient(pat\_nhs\_id, t)$	environment	pat_nhs_id: patient id, t: time of admission

ward_patient is an environmental predicate which takes two parameters, the patient identifier and the time of admission. We assume that this predicate can be checked at runtime, for example by querying a relation in a database, and that it reflects the current set of patients in the ward at any given time.

The policy could be expressed as:

$$doctor\_on\_duty\,(h\_id?),\ ward\_patient\,(pat\_nhs\_id?, t?) \vdash treating\_doctor(h\_id, pat\_nhs\_id).$$

The rules for parameter matching require that the value of the in-parameter h_id of $treating\_doctor(h\_id, y)$ must match the out-parameter h_id in the RMC for the prerequisite role *doctor_on_duty*. pat_nhs_id and t are output as the result of evaluating *ward_patient*, and the actual value of pat_nhs_id is also supplied as a parameter to the *treating_doctor* role and bound to the RMC that is generated. The parameter t in the predicate *ward_patient* is only a placeholder and its value would not be requested when querying the database.

In some situations, more than one binding may satisfy the rule, as is the case in the example above. Formally, this does not cause a problem; it simply means that any of those role instances may be activated within the session. It is therefore semantically correct to allow multiple outcomes, and in our example a new doctor coming on duty could acquire the *treating_doctor* role for each of the patients in the ward. Prolog would output all the possible answers in a similar situation. On the other hand, a user may often wish to activate a single role instance only, and it is therefore important to know how many tuples will satisfy a database query.

We can express the semantics of parameter matching very simply.

Definition 5.6 (*Parameter Matching*). If a variable u^t occurs in an activation rule $(x_1, x_2, \ldots, x_n \vdash r)$, then at activation time the same value of type t must be bound to each occurrence of u^t.

In the case that more than one set of bindings is consistent with the preconditions x_j, role activation succeeds with each set of variable bindings. Note that

all variables in the target role r occur in terms and therefore as in-parameters, so that each candidate set of variable bindings determines a unique RMC.

The notions of *initial rule* and *initial role* go over to the extended model without modification. We restate the definition here for clarity.

Definition 5.7 (*Initial Roles*). A role activation rule $(x_1, x_2, \ldots, x_n \vdash r)$, in which $x_j \in (\Omega \cup \mathcal{E})$ for all $1 \leq j \leq n$, is *initial*. The role r is said to be an initial role.

Example 3. Initial rules control user authentication at the start of a session. Note that any variable occurring as a parameter of the initial role must be bound, hence in such a case there must be at least one antecedent condition. For example, for a user to log in successfully the initial rule might include an environmental precondition $pwd(h\_id?)$ implemented by the program which checks the user's password and returns h_id as an out-parameter. The corresponding initial rule would be as follows:

$$pwd(h\_id?) \vdash local\_user(h\_id).$$

5.3 Environmental Predicates

Environmental constraints in the basic model allow dynamic aspects of a policy to be modelled. Since each is an atomic proposition their usefulness is limited. In the extended model, environmental constraints are renamed environmental predicates to emphasise that each may take a number of typed parameters. Since the parameters of environmental predicates may be of either mode they can be used in a variety of ways. Three primary uses are given below.

—specifying environmental constraints (much as in the basic model)
—acquiring context-sensitive information (for example by database lookup)
—setting attributes to a computed value

In EHR applications, access control decisions cannot be based on knowledge of the subjects and their privileges alone. Attributes such as the current time of day, patient assignment or the presence of particular fields in an EHR must also be taken into account. Other researchers have noted this requirement in connection with health records [Georgiadis et al. 2001; Giuri and Iglio 1997], and similar needs arise in active house applications [Covington et al. 2001]. It is therefore important to take context into account when authorizing access; in addition it is often helpful to do so when activating and deactivating roles. There are two main advantages: (1) it allows a natural modelling of real-world policies; (2) it becomes easier to specify privileges, since context-sensitive constraints can be enforced through the role activation rules. We illustrate these points with the help of the earlier examples.

In Example 1, we assume that a doctor can access the computer system at all times, but that she gains additional privileges during her duty hours. This is a natural example of an environmental constraint on access decisions; certain actions are only permitted to a doctor, and when she is on duty. The check could be enforced at each access, but that would carry an unnecessary overhead.

We therefore associate the extra privileges with the specific role *doctor_on_duty*, and use the environmental predicate *on_duty*(*h_id*) to control activation of that role. Note that it is natural to use a parameter, and also that parameter matching behaves as we should hope. In this case, the parameter value is obtained from the *local_user* RMC, and the environmental predicate then serves much the same purpose as in the basic model. It is vital that the implementation of *on_duty*(*h_id*) is active, also that it is marked as a membership condition in the role activation rule. Otherwise the doctor, if she remains logged in, could retain the extra privileges when she is no longer on duty. It is now easy to express privileges that are available to doctors only when on-duty. We believe that authorization policies expressed in terms of *on_duty* roles will be more readable, and that as a result fewer errors will be made.

In Example 2, we once again use an environmental predicate to extract context-dependent information, but in this case the mode of the parameters is *out*. Settings of the parameters *pat_nhs_id* and *t* will iterate over the possible combinations of patients and their admission times. In this way, it is easy to populate fields in a target RMC with values derived from a local database. If the same variable occurs as an out-parameter in more than one environmental predicate, the semantics is that of query evaluation with each shared variable serving as a join key. We expect that in most situations database lookup will be single-valued, and that policy managers will exploit key information in database schemas to ensure this.

Another use of environmental predicates is to allow computed values to be set in fields of a target RMC during role activation. This might be necessary in order to set a nonce value such as a session identifier to be used solely for audit purposes. In Example 3, we give a slightly different use, that of setting a parameter in an initial role. Because we require every variable occurrence to be bound it is essential to establish *h_id* as an out-parameter. This may seem rather artificial, but it has the great advantage of ensuring that all parameters are set explicitly.

Example 4. Let us return to Example 1 again. We did not discuss the validation of a doctor's medical qualification when activating the role *doctor_on_duty*, assuming that it was sufficient to check the locally issued appointment certificate *employed_medic*(*h_id*). A more realistic hospital-wide policy might be that *any person who is qualified by the British General Medical Council (GMC) and is in the hospital employee database can obtain a doctor role.* Suppose that the GMC issues an appointment certificate *qualified* to all qualified doctors; each certificate bears a unique sequence number *gmc_id* and also contains a number of attributes, such as surname, forename, date-of-birth, registration date and specialism. This certificate is intended for life-long use without any particular application in mind, it is merely an electronic assertion that the holder is a qualified doctor. To implement hospital policy it is necessary to match the information in a *qualified* certificate with a locally recognized username. We assume the information is held in the database *employee_db* which has a large number of fields, and we introduce an environmental predicate *emp_db_user* for this specific task. The environmental predicate *emp_db_user*

takes the unique sequence number *gmc_id* set by the appointment certificate *qualified* as an in-parameter, returning a Boolean to indicate whether the qualified doctor it identifies is a current employee; if the result is **true** it sets the out-parameter *h_id* to the local user id of the doctor. The appointment certificate *qualified* may have fields other than those shown.

Name	Type	Parameters
qualified(*gmc_id*, *sn*, *fn*, *d*, *sp*, ...)	appointment	*gmc_id*: identifier *sn*: surname *fn*: forename *d*: date of registration *sp*: specialism ...
emp_db_user(*gmc_id*, *h_id*)	environment	*gmc_id*: identifier *h_id*: local user id

We could now express this policy as below:

$$qualified(gmc\_id?, sn?, fn?, d?, sp?, \ldots),\ emp\_db\_user(gmc\_id, h\_id? \vdash\ doctor(h\_id).$$

Informally, this rule means that any person who can present a valid *qualified* certificate *and* is registered in the hospital's database can activate the role of *doctor*. The extra privileges associated with the specific role *doctor_on_duty* require a check of the additional environmental predicate *on_duty*(*h_id*). We discuss how the *qualified* appointment certificate might be validated in Section 6.2.

An alternative approach would be to implement a unary function *gmc_user* to convert from a *gmc_id* to the corresponding local *h_id*, raising an exception if the GMC certificate does not identify a local doctor. Type checking will ensure that the function *gmc_user* sets an appropriate value in the *doctor* RMC.

$$qualified(gmc\_id?, sn?, fn?, d?, sp?, \ldots) \vdash\ doctor(gmc\_user(gmc\_id)).$$

As explained in this section, the power of environmental predicates is greatly enhanced by including parameters. They increase the expressive power of the model without adding unnecessary complexity. In particular, their use offers a uniform way of introducing information from local databases, as well as supporting context-sensitive behaviour such as temporal constraints.

5.4 Validity Rules for Appointment Certificates

In Definition 4.3 of the basic model, we described how the issuer of an appointment certificate can restrict its use by defining a set of prerequisite roles and a set of environmental constraints. In this section, we extend that definition to include parameters.

Definition 5.8 (*Validity Rules*). The issuer of an appointment certificate can define a *validity rule* that must be satisfied whenever the appointment certificate is presented to meet a precondition during role activation. The syntax

for specifying validity rules in an appointment certificate bears a close resemblance to the definitions of Section 5.2, except that there is no target role. A validity rule has syntax:

$$(x_1, x_2, \ldots, x_n \vdash)$$

where each x_j for $1 \leq j \leq n$ (*a validity condition*) is a predicate expression derived from some predicate symbol in the universe $(\mathcal{R} \cup \mathcal{E})$.
We define additional syntactic constraints on the parametrizations:

—If a validity condition $x_j = \rho(u_1, \ldots, u_n)$ for some role predicate symbol $\rho \in \mathcal{R}$, then each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a value c of type t_i.
—If a validity condition $x_j = \epsilon(u_1, \ldots, u_n)$ for some predicate symbol $\epsilon \in \mathcal{E}$, then each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a term of type t_i.

As in the case of role activation rules there must be no free variables. If a variable occurs as an in-parameter within some validity condition, then it must also occur as an out-parameter in some other validity condition of the same rule. Validity conditions are interpreted by the issuer when the appointment certificate is presented for validation during role activation, for more details, see Section 5.6. Each condition may optionally be specified as *active*, in which case it has the same status as a membership condition in a role activation rule. We discuss this further in Section 5.7.

5.5 Privileges and Authorization Rules

So far, we have not discussed how privileges are derived from roles in the extended model. In the basic model, a relation RP is defined to describe the role-privilege relationship. In the extended model, we must take role parameters into account at the time of access. Access decisions may also be context-sensitive. We therefore need a more flexible approach to privilege assignment than can be derived from a simple relation.

A privilege in the extended model takes the form of a predicate symbol for which each parameter has some specified type. The binding to concrete privileges depends on the application context, but we assume that the rights conferred by an instantiated privilege depend on the actual values of its parameters.

Definition 5.9. A privilege instance is an n-ary predicate expression $\pi(c_1, \ldots, c_n)$, where $\pi \in \mathcal{P}$ is a predicate symbol of signature $(t_1 \times \cdots \times t_n) \rightarrow \mathit{bool}$ and for each $1 \leq i \leq n$, c_i is a value of type t_i.

As an example, one could define a privilege instance as follows:

Privilege instance	Informal description of right
$\mathit{read\_EHR}(y, f)$	read field f from the electronic health record of patient y

The definition of privileges and their bindings is entirely application-dependent. With the use of parameters it is possible for an application to

establish rights at a fine-grained level, such as method invocation between distributed services. The level of granularity depends on the security policy and implementation. Our formalism neither restricts nor encourages enforcement at a particular level.

Privilege assignments are specified through *authorization rules*. An authorization rule associates a privilege with an *authorizing role*. Authorization rules take exactly one role on the left-hand side and may optionally have a number of environmental predicates as *authorizing conditions*. The intention is to provide active security by considering the context of a request at access decision time. The syntax is similar to that of activation rules and validity rules.

Definition 5.10 (*Authorization Rules*). An *authorization rule* is defined as a sequent

$$(r, e_1, \ldots, e_n \ \vdash \ p),$$

where $r = \rho(u_1, \ldots, u_n)$ for a role symbol $\rho \in \mathcal{R}$, each $e_j = \epsilon_j(u_1, \ldots, u_n)$ for some $\epsilon_j \in \mathcal{E}$ and $p = \pi(u_1, \ldots, u_n)$ for some $\pi \in \mathcal{P}$. We say that r is the authorizing role instance, each e_j is an authorizing condition and p is the target privilege instance. We define additional syntactic constraints on the parametrizations.

—In the authorizing role instance r, each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a value c_i of type t_i.

—In each authorizing condition e_j, each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a term of type t_i.

—In the target privilege instance p, each $u_i^{t_i}$ must be either an out-parameter $y^{t_i}?$ or a value c_i of type t_i.

CONSTRAINT 2. *There must be no free variables in an authorization rule*

$$(r, e_1, \ldots, e_n \ \vdash \ p).$$

Note that the treatment of parameters on the right-hand side of the rule is quite different from that in role activation rules, where the aim is to instantiate the fields of a newly created RMC. In this case, we know the details of both the authorizing role instance and the access that has been requested, and we must make a decision in the current context. A rule is applicable only if the fields c_i match; in that case, any out-parameters associated with the authorizing role and the target privilege are set, then the authorizing conditions are evaluated. A case of interest is that of a variable whose only occurrence is as an out-parameter on the right-hand side of the rule. The decision on whether to authorize access is independent of the value of the parameter in that position, effectively supporting universal quantification over target privilege instances.

The use of authorization rules is demonstrated in the following example:

Example 5. We continue the medical Examples 1 and 2 given in Section 5.2. Suppose that a *treating_doctor* in the ward is allowed to read fields 1, 3 and 4 only from the EHR of a patient under her care. A possible way of specifying this policy is to extend the system with the following components:

Name	Type	Meaning and parameters
read_EHR(*y*, *f*)	privilege	read a field from a patient's EHR *f*: field name *y*: patient identifier
check_field_td(*f*)	environment	check within rights of treating doctor *f*: a field name

We introduce a new environmental predicate *check_field_td* to check whether a particular field in a patient's record may be read by the doctor in charge of the case. This predicate takes a field name f from the access request and returns **true** if f is either 1, 3 or 4, and **false** otherwise.

We can now express the authorization policy:

$$\mathit{treating\_doctor}(x?, y?),\ \mathit{check\_field\_td}(f) \ \vdash\ \mathit{read\_EHR}(y?, f?).$$

Suppose that a doctor Alice is providing treatment for a patient Bob. She presents the authorizing role instance *treating_doctor*(Alice, *Bob*) with her request to read field 3 of Bob's EHR, and the variables x, y and f are set to Alice, Bob and 3 respectively. Calling the environmental predicate *check_field_td* confirms that Alice as treating doctor may access field 3 of patient Bob's EHR.

Note that in this case the privilege instance identifies the object of the access, a particular field in Bob's EHR; it is not necessary to identify the subject requesting the access, Alice, who has been authenticated into the role *treating_doctor*, with Bob a patient under her care.

5.6 Model Semantics

In the basic model, the interpretation of activation rules in a particular context is defined by a truth assignment function, which establishes what it means for each precondition to be satisfied. In the extended model, we must take account of parameters, and also handle the term algebra. We need an *interpretation* that assigns a concrete object of appropriate type to each syntactic element and we must then define *satisfaction* based on this assignment. The treatment applies equally to role activation rules, validity rules and authorization rules.

In the discussion that follows, we first show how parameter binding is managed within the framework of a general rule. The only tricky case is that of environmental predicates, which can contain both in- and out-parameters. We show how to model environmental predicates by environment queries, which update a table of variable bindings specific to the rule that is being interpreted. The order of evaluation of environmental predicates may be determined statically within any given rule. We next show how within a specific context to evaluate terms that contain only constants and function symbols; we are then in a position to interpret each predicate expression as a concrete instance of some element of the access control system, and thus determine whether a particular rule is satisfied. The interpretation developed in this section therefore deals with:

—Interpreting environmental predicates as queries

—Parameter binding and the order of evaluation of predicates

—Term evaluation

—Interpreting predicate expressions
—Validity rules for appointments
—Role activation rules
—Authorization rules.

The interpretation is based on the presentation in terms of propositions developed in Section 4.3 in the sense that we evaluate each rule within a *snapshot* of the system. For an implementation based on active platforms, this snapshot semantics is sufficient, since it gives a strong guarantee of the state of a system at each decision point, and valid state transitions are enforced by active notification. The details of the design of an event platform that will support this semantics can be found in Bacon et al. [2000].

Definition 5.11 (*Interpretation*). An interpretation with respect to a user session u, I_u, is a triplet (D, U, M), where:

—D is a nonempty set of typed values, called the *value domain*. The subset of D of all values of type t is denoted D_t.
—U is a nonempty set of access control system entities, called the universe. Distinct subsets of U correspond to role instances, appointment certificates, environmental queries and privilege instances.
—M is a mapping from syntactic elements to typed values, functions or access control system entities, which assigns a meaning to each element of a rule.

The initial implementation protects SOAP services, and values have an associated XML/Schema type. There is also an essentially complete Java implementation.

5.6.1 *Environment Queries.* Rules of all three kinds may contain environmental predicates. In each case, other predicates in the rule may contain either in- or out-parameters, but **never** both. Before an environmental predicate is evaluated, the value of any in-parameter must be bound; the process of evaluation returns (a sequence of) sets of bindings for the out-parameters. Suppose that an environmental predicate expression $e = \epsilon(u_1, \ldots, u_n)$ for some predicate symbol $\epsilon \in \mathcal{E}$, where each $u_i^{t_i}$ is either an out-parameter y^{t_i} ? or a term of type t_i. We assume that the value of each in-parameter is known, hence each term can be evaluated. The predicate can then be decided by a query that applies a filter specifying the value of each known parameter (cf. a **WHERE** clause in SQL); the query returns all possible sets of bindings for the out-parameter positions such that the predicate evaluates to **true**. Our extension to PostgreSQL supports *active queries*, which have the property that the requester is notified if the result should subsequently change.

Definition 5.12 (*Environment Query*). An environment query $E \in U$ is an implementation of an environmental predicate $\epsilon \in \mathcal{E}$. If the signature $\Sigma(\epsilon) = (t_1 \times \cdots \times t_n) \rightarrow bool$, then E will accept filter expressions that specify correctly typed values for some subset of the arguments, and return as result the possible extensions for which the predicate holds.

In a practical implementation of an environmental predicate, only certain subsets of the arguments will define valid filter expressions. In particular, it is possible to ensure that if a variable binding exists, then it is unique. Some predicates may be valid only if every parameter is an evaluated term, for example the binary predicate LT_t which compares the order of two values of type t. We do not go into details.

5.6.2 *Parameter Binding and the Order of Evaluation of Predicates.* Suppose that a rule takes the form:

$$(x_1, x_2, \ldots, x_m, e_1, \ldots, e_n \vdash p),$$

where each x_j for $1 \leq j \leq m$ is a predicate expression derived from some predicate symbol in the universe $(\mathcal{R} \cup \Omega)$, and each e_i for $1 \leq i \leq n$ is a predicate expression derived from a predicate symbol in the universe $\mathcal{E}$.

We require that there are no free parameters in the rule. The target predicate expression p contains only in-parameters in a role activation rule, and only out-parameters in an authorization rule. Each of the predicate expressions x_j also contains only out-parameters. Variable binding is therefore straightforward with one exception. We first bind all out-parameters associated with preconditions x_j or with a target privilege expression. Once binding is complete we can set the parameters in a target role expression. In what order should we evaluate the environmental predicates?

Note that once a particular variable is bound to some value any later binding of the same variable must yield the same value; if an earlier evaluation was multi-valued we can backtrack, otherwise the predicate is **false**. Our aim is to order the evaluations of $\{e_i \mid 1 \leq i \leq n\}$ so that whenever a variable occurs as an in-parameter it has already been bound. The algorithm is similar to a topological sort, with the slight complication that a variable may have more than one occurrence as an out-parameter among environmental predicate expressions. The rule parser should reject as ill-formed any rule in which there are cyclic dependencies between the parameter occurrences.

In this way, each rule stored at a service may be annotated with the (partial) order of predicate evaluation. During interpretation of the rule we maintain a list of variable bindings. When interpreting an environmental predicate expression, we use the current list to set up the filter expression for the corresponding environment query. If there are out-parameters to be set, the filter query yields a list of additional variable bindings that satisfy the predicate; provided there is no conflict the list of variable bindings is extended and evaluation moves to the next predicate. In the case of conflict, we first look for a new set of bindings for the same filter; otherwise, we backtrack if possible. In what follows, we assume that before interpreting a predicate expression we have evaluated all the terms.

The rest of this section mirrors the organization of Section 5.1, in that we show how to interpret each syntactic element and hence give a complete account of what it means for a rule to be satisfied.

5.6.3 *Term Evaluation.* We first define the meaning of typed terms in the many-sorted algebra. The bindings of variables are established through the implementation, and the meaning of an in-parameter that occurs in a term is therefore an element of the value domain.

—If c is a value of type t, $M[c] \in D_t$.
—If u_i^t is an in-parameter of type t that is bound to value d_t, $M[u_i^t] = d_t \in D_t$.
—If f is an n-ary function symbol with $\Sigma(f) = (t_1 \times \cdots \times t_n) \to t$, $M[f] \in (D_{t_1} \times \cdots \times D_{t_n}) \to D_t$.

The implementation $M[f]$ of a function symbol f may be context-sensitive. In particular, a 0-ary function may not yield the same value in every context, consider for example *current_user* and *current_time*. The meaning of each term can now be defined in the obvious way.

5.6.4 *Interpreting Predicate Expressions.* If $\phi \in (\mathcal{R} \cup \Omega \cup \mathcal{P} \cup \mathcal{E})$ is a predicate symbol of signature $(t_1 \times \cdots \times t_n) \to bool$ and $u_1, \ldots, u_n$ are expressions of types $t_1, \ldots, t_n$, then $\phi(u_1, \ldots, u_n)$ is an n-ary predicate expression.

If $\phi \in \mathcal{R}$, $M[\phi] \in U$ identifies some role of the access control system that we are modelling. Parameter values correspond to parameters in the associated RMCs.

If $\phi \in \Omega$, $M[\phi] \in U$ identifies some appointment type of the access control system. Parameter values correspond to parameters in the associated appointment certificates.

If $\phi \in \mathcal{P}$, $M[\phi] \in U$ identifies some privilege of the access control system. Parameter values identify a concrete privilege instance.

If $\phi \in \mathcal{E}$, $M[\phi] \in U$ identifies the corresponding environment query. We have already explained how parameters and variable bindings will be managed in this case.

We next consider what it means for a particular predicate expression to be satisfied within some rule. We have already described the order of evaluation, and can assume that evaluation takes place in the context of a list of bindings of variables v^t of type t to values $d_t \in D_t$, and that the rule checker can inspect the access control environment. We do not consider the mechanics by which RMCs and appointment certificates are made available during role activation and service invocation.

Prerequisite RMCs, appointment certificates and privilege instances are all available during rule checking as concrete tuples of type $(D_{t_1} \times \cdots \times D_{t_n})$ say. The only parameter types permitted in a corresponding predicate expression defined in a rule are out-parameters and constant values of the appropriate type. We define what it means for such a predicate to be satisfied within a particular interpretation.

Definition 5.13 (*Predicate Satisfaction).* Given an interpretation $I_u = (D, U, M)$ with respect to a user session u and an n-ary predicate expression $\Phi = \phi(u_1^{t_1}, \ldots, u_n^{t_n})$, we say that I_u satisfies Φ, written as $I_u \models \Phi$, if there is a concrete instance of the access control entity $M[\phi]$ whose fields $(d_1, \ldots, d_n)$ are such that whenever $u_i^{t_i} = c^{t_i}$ is a constant of type t_i, then $M[c^{t_i}] = d_i$.

When during the interpretation of some rule such a predicate expression Φ is satisfied, the variable list is updated for each out-parameter $u_j^{t_j} = y^{t_j}?$ by binding variable y to value d_j. If this binding produces conflict, then rule interpretation backtracks if possible; otherwise, the rule is not satisfied in the interpretation I_u.

It is now easy to define satisfaction for rules of each kind. For simplicity, we begin with the case of a validity rule for an appointment certificate.

5.6.5 *Validity Rules for Appointments.* The issuer of an appointment certificate can restrict the contexts in which its use is valid by specifying both prerequisite roles and environmental predicates. The form of a validity rule is therefore:

$$(x_1, x_2, \ldots, x_m, e_1, \ldots, e_n \vdash),$$

where each x_j for $1 \leq j \leq m$ is a predicate expression derived from some role symbol in the universe $\mathcal{R}$, and each e_i for $1 \leq i \leq n$ is a predicate expression derived from an environmental predicate $\epsilon_i \in \mathcal{E}$.

Note that the validity rule is specified explicitly by the issuer and forms an extension of the appointment certificate, hence the values of any fields in the validity rule are bound at the time of issue. During role activation the appointment certificate is sent to the issuer for validation; in addition to checking the signature the issuer ensures that the validity conditions are satisfied. Note that the naming of roles and environmental predicates is interpreted within the issuer's context.

Validation of the conditions is straightforward. First, ensure that each of the prerequisite role conditions x_j is satisfied according to Definition 5.13. The check includes validating the RMC with the issuing service. Assuming that there has been no conflict of bindings we derive a list of variable bindings in whose context the environmental predicate conditions e_i can be verified through a sequence of environment queries, see Definition 5.12. The validity rule of the appointment certificate is satisfied if each validity condition is satisfied.

The issuer can specify that some subset of the validity conditions are membership conditions. We discuss the semantics in Section 5.7.

5.6.6 *Role Activation Rules.* We can now define the semantics of role activation rules. The syntax of an activation rule is as follows:

$$(x_1, x_2, \ldots, x_\ell, y_1, y_2, \ldots, y_m, e_1, \ldots, e_n \vdash r),$$

where each x_j for $1 \leq j \leq \ell$ is a predicate expression derived from some role symbol in the universe $\mathcal{R}$, each y_k for $1 \leq k \leq m$ is a predicate expression derived from some appointment symbol in the universe Ω, and each e_i for $1 \leq i \leq n$ is a predicate expression derived from an environmental predicate $\epsilon_i \in \mathcal{E}$. r is the target role expression.

The activation rule is satisfied whenever there is a list of variable bindings with respect to which each of the predicate expressions on the left-hand side evaluates to **true**. We defined what it means for a role condition to be satisfied in the previous section. For an appointment condition we present the appointment

certificate to the issuer for validation; the issuer checks the signature, and also verifies the validity conditions (if any). If each of these conditions is satisfied, we can proceed to check the environmental predicate conditions exactly as for a validity rule. The role activation rule is satisfied if each of the preconditions is satisfied.

When the role activation rule is satisfied, each of the variables occurring on the left-hand side is bound. Each of the terms defining a parameter of the target role expression can therefore be evaluated. The target RMC is instantiated and returned to the user.

5.6.7 *Authorization Rules.* The interpretation of authorization rules differs in one important respect. When a request is made to a service the access control monitor determines what specific privilege instance, say $P(d_1, \ldots, d_n)$, the user requires (see Section 5.5). This privilege instance, the *target privilege*, is regarded as the goal. If $P = M[\pi]$, then it is essential to find an authorization rule of the form

$$(r, e_1, e_2, \ldots, e_m \ \vdash \ p),$$

where $p = \pi(u_1, \ldots, u_n)$. If I_u satisfies such a rule for some authorizing role instance held within the user session u, then the access is authorized and the request proceeds.

The authorization rule is satisfied whenever there is a list of variable bindings with respect to which each of the preconditions evaluates to **true**, and such that if an out-parameter $y?$ appears on the right-hand side as u_j, then y is bound to the corresponding value d_j in the target privilege. We check whether each of the conditions on the left-hand side is satisfied exactly as we did for a validity rule; the order of evaluation of the predicates is established by the parser at the time the rule is defined.

5.7 Role Deactivation

Each role activation rule can specify a set of membership conditions whose enforcement is active, see Section 5.2. If a membership condition is a prerequisite role, then an event channel will be set up when the RMC is validated, and the activating service will be notified should the prerequisite role be relinquished. For environmental predicates, the implementation is equivalent; an environment query E is executed during role activation, and an event channel is established to notify the activating service of any change.

For appointment certificates, the situation is complicated by the validity rules. An appointment certificate that is **not** a membership condition need only be valid at activation time; the issuing service checks the signature and enforces the validity rules, but there is no need to establish an event channel. For membership conditions, we need an event channel, since the picture may change in two ways. First, the appointment certificate may be revoked; the revocation is flagged in the persistent data structure recording its issue, and sessions in which it has served as a membership condition are notified. Second, there may be one or more active validity conditions associated with its continuing use. When the issuing service checks an active validity condition an event

channel is established; should the condition become **false** the issuing service is notified, and then in turn the service that presented the appointment certificate for validation. Note that in the latter case the appointment certificate remains valid, and the persistent data structure is not altered.

6. CASE STUDIES

6.1 Support for Anonymity

Suppose that privacy legislation has been passed whereby someone who has paid for medical insurance may take certain genetic tests anonymously. The insurance company's membership database contains data about policy holders; the genetic clinic has no access to this and the insurance company may not know the results of the genetic test, or even that it has taken place. To meet these requirements, it is essential for the system to refer to an individual by a pseudonym that cannot be linked to the identity of the individual. In OASIS, it is possible to use a role as a pseudonym so long as there are no links back to the identity of its holder. In the clinic, we define a role *paid_up_patient*.

Name	Type	Meaning
paid_up_patient	role	authenticated patient to the clinic

The clinic, for accounting purposes, must ensure that the test is authorized under the scheme. Depending on the clinic's policy, this can be done manually or automatically. We assume that a member of the insurance scheme is issued with an anonymous membership card showing the expiry date. If the clinic follows the manual approach, the card is checked manually and a role *paid_up_patient* is assigned to the patient. This assignment is expressed as an initial role

$$\vdash \ paid\_up\_patient.$$

For greater security, the insurance scheme membership card could contain a computer-readable appointment certificate and expiry date. The clinic's authentication process can then be automated to use appointment and environmental predicates, such as:

Name	Type	Meaning
$insurance\_membership(\ldots, t, \ldots)$	appointment	t: expiry date
$LT\_time(t1, t2)$	environment	$t1, t2$: timestamps

The appointment certificate *insurance_membership* may contain a number of parameters, one of which is the expiry time of the membership. The environmental predicate *LT_time* compares two parameters of type *timestamp*. The policy could then be expressed as follows:

$$insurance\_membership\,(\ldots, t?, \ldots),\ \ LT\_time(current\_time, t) \vdash \ paid\_up\_patient.$$

A patient can become active in the role *paid_up_patient* if and only if her membership of the insurance company is valid, which is proved by validating the

appointment certificate and checking whether the membership has expired. Note that an instance of the active role *paid_up_patient* cannot be used to trace back the identity of the patient, since all such instances are identical from the system's perspective.

6.2 Multidomain Healthcare System

As indicated in Section 5, a distributed electronic health record (EHR) system has been a motivating case study during the deployment of OASIS. Some of the engineering issues for this system have been discussed in Bacon et al. [2001b]; here we will model a simplified subset of the policies.

We assume that there are a number of interacting domains, including the General Medical Council (GMC) domain, a national EHR domain and many hospitals and research institutes. Each domain is autonomous and has the authority to define its own roles, appointments and policies. Domains may form cooperative relationships by entering service-level agreements (SLAs) that define standards for interoperation. SLAs will typically involve a set of roles, appointments, environmental predicates and privileges, and will indicate how these entities are to be used in services' activation and authorization rules. SLAs may also define attributes such as degree of trust, responsibilities and liabilities but we shall not go into these aspects here.

The GMC is the official body for regulating doctors' qualifications. We assume that the GMC domain issues a *qualified* appointment certificate for every doctor who meets their academic and professional requirements. This may be stored in a smartcard and is intended to have a life-long duration. It will contain basic information such as the name, date of certification and specialism of the doctor together with a unique identifier *gmc-id*. The GMC provides an environmental predicate *not_revoked* for checking the validity of the qualification identified by *gmc_id*. Both *qualified* and *not_revoked* form part of the SLA for interworking with the GMC domain and are available in every healthcare domain. A summary follows:

Name	Type	Meaning
qualified(*gmc_id*, *sn*, *fn*, ...)	appointment	Certify the named individual (*sn*, *fn*) (surname, forename) as a doctor. The certificate has a unique identifier *gmc_id*.
not_revoked(*gmc_id*)	environment	Returns **true** if the certificate with identifier *gmc_id* has not been revoked, or **false** otherwise.

The national EHR domain consists of a reliable, replicated EHR service, which maintains a virtual health record for each individual. A health record comprises an index of the treatment history of that patient together with a nonconfidential header, including name and address, and emergency information such as blood group, allergies and current medications. We make use of a

subset of access control policies for health records which are likely to be specified by law:

—A doctor of a patient may view the header of the patient's EHR.
—A doctor treating a patient may submit the record of treatment to be appended to the patient's EHR.
—Particular individuals may be excluded from accessing a patient's EHR. This could create an exception to the general policy and in this case will override it.

To implement these policies, the national EHR domain defines a SLA, to which all healthcare domains that wish to access patient records must comply. The SLA requires interoperating domains to export a role *local_doctor*(*doc_nhs_id*, *pat_nhs_id*) to identify the treating doctor *doc_nhs_id* of patient *pat_nhs_id*. Both are national health-service identifiers. Patients can express a wish to exclude particular individuals from accessing their EHRs; this information is transmitted to the national EHR domain which maintains a database. Exclusion lists are implemented by consulting this database. Database lookup is abstracted using an environmental predicate *not_excluded*(*owner*, *requester*), which returns **true** if the *requester* is not explicitly excluded from accessing the EHR of *owner* and **false** otherwise.

The operations available to clients include *get_header* and *append_treatment*, which are expressed by abstract privileges with the same names. An abstract privilege may not always have an identical form to the actual method invocation; for example, a request to *append_treatment* will submit the details of treatment as well as parameters such as *doc_nhs_id* and *pat_nhs_id*, but only the last two are needed to identify the specific privilege instance. The definitions of these components of the EHR domain are summarized below:

Name	Type	Meaning
not_excluded(*owner*, *requester*)	environment	Returns **true** if *requester* is not restricted from accessing the EHR of *owner*.
get_header(*doc_nhs_id*, *pat_nhs_id*)	privilege	Retrieve header of the EHR of patient *pat_nhs_id* for doctor *doc_nhs_id*
append_treatment(*doc_nhs_id*, *pat_nhs_id*)	privilege	Submit details of treatment of patient *pat_nhs_id* by doctor *doc_nhs_id*.

We can now specify the authorization rules for the EHR domain.

$$@.local\_doctor(doc\_nhs\_id?, pat\_nhs\_id?),\ not\_excluded(pat\_nhs\_id, doc\_nhs\_id) \vdash get\_header(doc\_nhs\_id, pat\_nhs\_id) \quad (1)$$

$$@.local\_doctor(doc\_nhs\_id?, pat\_nhs\_id?),\ not\_excluded(pat\_nhs\_id, doc\_nhs\_id) \vdash append\_treatment(doc\_nhs\_id, pat\_nhs\_id) \quad (2)$$

We refer to components defined in other domains using a dot-notation, for example *domain.rolename*, with the @ symbol meaning *any* domain. The interpretation of *any* is domain-specific; in this case the EHR service might check the originating domain against a list of participating medical domains.

A hospital is a domain comprising several services such as pharmacy and X-ray and, in particular, a local EHR service for interacting with the national EHR service. Suppose that the hospital has a prior agreement with both the GMC and the national EHR domains. The hospital issues an $employed(h\_id)$ appointment certificate for each member of staff with a hospital identifier h_id. This is used as a credential for activating roles in the hospital. In practice, this may be contained in a smartcard or a name tag carried by all staff. Before hiring a new doctor, the hospital checks the applicant's medical qualifications with the relevant authority, typically the GMC for those who trained in the UK. An entry is made in a database to record the gmc_id of the new doctor, and this can be recovered by the environmental predicate $map\_gmc\_id(h\_id, gmc\_id)$. By agreement, the GMC domain includes an environmental predicate $not\_revoked(gmc\_id)$ that checks that the doctor identified has not been struck off. For extra security, this check may be (part of) the validity rule for the *employed* appointment certificate issued to each doctor:

$$GMC.not\_revoked(GMC\_ID) \vdash . \quad (3)$$

This rule is associated with the particular *employed* appointment certificate issued to the new doctor, whose GMC serial number is known at the time of issue. We expect a similar validity rule to be defined for an appointment certificate issued to a qualified nurse. We assume that an *employed* appointment certificate captures the nature of the employment, as a doctor, nurse, consultant in some specialism, etc. This might be in the form of additional parameters in a generic employment certificate or by having differently named certificates for different categories of employees.

A doctor coming on duty may activate a role $doctor\_on\_duty(h\_id, dept)$, where h_id is the doctor's hospital identifier and *dept* is the department to which she is assigned. The activation rule for this would be:

$$employed(h\_id?),\ is\_doctor(h\_id, dept?)^* \vdash doctor\_on\_duty(h\_id, dept). \quad (4)$$

We use an environmental predicate *is_doctor* in the above rule, implemented as an environment query which checks that the user h_id is a doctor and if so sets variable *dept* to the department to which she is assigned. As a notational shorthand, we denote membership conditions by asterisks. Suppose that in a particular case a doctor with hospital identifier H_ID comes on duty in the

A&E department, indicated by *AE*. In activating this rule, *is_doctor*(*H_ID*, *AE*) becomes a membership condition for the RMC *doctor_on_duty*(*H_ID*, *AE*).

In an A&E department, there is a specific policy under which a nurse on-duty may become a screening nurse responsible for checking in new patients and assigning each to a doctor. Assignment of a patient could be handled through an appointment *AE_patient*, with the appointer role being *screening_nurse*. It carries two parameters, the local identifier *h_id* of the doctor assigned and the national health service ID of the patient, *pat_nhs_id*. Once assigned to a patient, a doctor on duty in A&E may activate the *treating_doctor* role, here extended with an additional parameter *dept* to indicate the department, in this case *AE*.

$$\begin{aligned} &\mathit{doctor\_on\_duty}(h\_id?, AE)^*,\ \ \mathit{AE\_patient}(h\_id, pat\_nhs\_id?)^* \\ &\qquad \vdash\ \mathit{treating\_doctor}(h\_id, pat\_nhs\_id, AE) \end{aligned} \tag{5}$$

Note that the *-denoted conditions are membership conditions, that is, according to this policy the *treating_doctor* role is revoked when the doctor goes off-duty and/or the patient assignment is withdrawn.

Consider a scenario where a patient arrives in A&E and has been assigned to a doctor after being screened. The treating doctor needs to refer to the emergency information in the patient's EHR and makes a request to the EHR service in the local hospital domain with a credential asserting membership of the role *treating_doctor*. Because the SLA with the national EHR requires a role *local_doctor* to be exported, the local EHR service maps an instance of *treating_doctor* into *local_doctor*, expressed in the following activation rule:

$$\begin{aligned} &\mathit{treating\_doctor}(h\_id?, pat\_nhs\_id?, dept?),\ \ \mathit{map\_nhs\_id}(h\_id, doc\_nhs\_id?) \\ &\qquad \vdash\ \mathit{local\_doctor}(doc\_nhs\_id, pat\_nhs\_id). \end{aligned} \tag{6}$$

No mapping is required for the patient identifier, as we have used the patient's health-service identifier throughout. The *dept* parameter is a place holder, of no interest at national level. Having mapped the role (by activating it), the local EHR service acts as an agent and sends the request to the national service. Access would be authorized by rule (1) if the treating doctor is not on the patient's exclusion list.

Suppose that the treating doctor decides to order some medication for the patient. She would first need to check the patient's allergies and current medication. This information is available in the emergency part of the EHR, which has already been fetched. The doctor will appoint someone to collect the drugs from the pharmacy. The details of the drugs, authorizer and collector are recorded in the pharmacy database for audit.

The doctor must at once record the new medication in the EHR header so that the emergency information is up-to-date. Also, when treatment is complete, she will submit the record of treatment to be appended to the patient's EHR. The process is similar to that of retrieving an EHR; a request is made to the local EHR service in the hospital domain, which acts as an agent for the requesting doctor to the national EHR service.

The components we have defined in this hospital domain example are shown in Figure 1 for reference.

Name	Type	Meaning
employed(*h_id*)	appointment	Proof of employment of local user *h_id*.
AE_patient(*h_id*, *pat_nhs_id*)	appointment	A&E patient *pat_nhs_id* is assigned to doctor *h_id*.
map_gmc_id(*h_id*, *gmc_id*)	environment	Maps between a local user id *h_id* and a GMC serial number *gmc_id*.
map_nhs_id(*h_id*, *doc_nhs_id*)	environment	Maps between a local user id *h_id* and a national health-service (NHS) id *doc_nhs_id*.
is_doctor(*h_id*, *dept*)	environment	Returns **true** if user *h_id* is a doctor assigned to department *dept*; **false** otherwise.
doctor_on_duty(*h_id*, *dept*)	role	User *h_id* is a doctor on duty in department *dept*.
treating_doctor(*h_id*, *pat_nhs_id*, *dept*)	role	Doctor *h_id* is treating patient *pat_nhs_id* in department *dept*.
local_doctor(*doc_nhs_id*, *pat_nhs_id*)	role (public)	A doctor treating a patient, each identified by an NHS id. Complies to the SLA of the national EHR.

Fig. 1. Definitions of the components in a hospital domain.

7. RELATED WORK

There have been a number of proposals to extend RBAC to meet the stringent requirements of *active security*. This means including context in some way in order that access control decisions can be based on attributes in addition to simple subject/object pairs. We compare OASIS with temporal access models, team-based models, content-based access control and generalized role-based access control models. OASIS can meet most of the requirements that these models address in a conceptually uniform manner through its support for parametrization and application-defined environmental predicates. The logic-based approach that we adopt allows a level of static verification. We believe that such an approach is vital in order to maintain the consistency of policies within large organizations and across distributed systems.

Bertino et al. proposed a temporal extension to RBAC called Temporal RBAC (TRBAC) [Bertino et al. 2000] based on their earlier work on temporal authorization in database systems [Bertino et al. 1996]. TRBAC introduces periodic activation and deactivation, and role triggers for expressing temporal dependencies. Periodic activation and deactivation support time-limited authorization. In OASIS, such requirements can be met by using environmental predicates to test *timestamps* in authorization and membership rules. There is no notion equivalent to TRBAC's role triggers in OASIS. Role triggers in TRBAC are used to express temporal inter-role dependencies, which enable a chained combination of activations or deactivations. In OASIS, we support a more general form of inter-role dependency based on activation and membership conditions. A similar effect to role triggers could be achieved in OASIS with some additional implementation effort, subject to the proviso that a role can only be activated within an authenticated session. We believe however

that the OASIS model on its own is sufficiently powerful for most applications, and such an extension would rarely be needed.

An abstraction of the context of an application can be expressed through the notion of teams, see the work of Thomas et al. on Team-Based Access Control (TMAC) [Thomas 1997; Georgiadis et al. 2001]. A team is a group of users in specific roles, collaboratively working on a common task. Contexts in TMAC are directly associated with a team, and the privileges that a user has are determined by her current team. This is appropriate if a task that requires collaboration is performed repeatedly, so that it makes sense to specify the privileges of a team that can carry it out, identifying the responsibilities of each individual member. In TMAC, the privileges that a user possesses are those directly assigned to her roles, combined with those of her team members using an operator (e.g. union), filtered by some context of the team. It is thus possible for a user to gain privileges through her teammates beyond those that she needs for her part in the task. Although this can be prevented by tightening the filtering context to identify individual team members, this defeats the purpose of RBAC. In OASIS, privileges are authorized by rules which can test application specific environmental predicates, and we could handle team membership in this way. Since activation and authorization rules have a uniform logical structure it is possible to reason about the propagation of privileges, at least in principle. In practice, such analysis will succeed only to the extent that environmental predicates are decidable.

Interesting research that is closely related to OASIS is the work on content-based access control described in Giuri and Iglio [1997]. They propose parametrization of roles through the use of role templates. Although their parametrization technique appears similar to ours, there are a number of crucial differences. Their work is based on extending privileges to include parametrized constraints that are evaluated at access time. A role is regarded as a set of privileges, so the parameters of a role template correspond to those of its privileges. In OASIS, roles are organizational entities, corresponding to specific tasks or positions. Parameters of a role are selected because they are likely to be useful when expressing policy. Indeed, a role may have apparently redundant parameters, not used directly within the issuing domain. Such parameters might be included to allow for more sophisticated policies in some future release. In Giuri's model, the constraints of a privilege are evaluated against the content of the requested object and the requesting subject only. OASIS, on the other hand, allows access decisions to be based on a combination of subject and application-defined environmental predicates, which may include object attributes, temporal conditions, or any other attributes. Our model is more flexible and expressive than content-based access control, since role activation can itself be made conditional on environmental predicates. OASIS also supports the notions of session and appointment which are absent from Giuri's model.

OASIS is designed to be general and flexible in order to support a wide variety of active applications, unlike most other contextual access control models. An exception is Generalized RBAC (GRBAC) [Covington et al. 2000, 2001], which is also generic. GRBAC extends traditional RBAC by introducing object roles and environment roles in addition to subject roles. An object role

represents a facet of the requested object, for example, date of creation; an environment role represents some environmental condition at the time of access. These roles are activated automatically by the system. We believe that the use of the term *role* for these purposes is counterintuitive, and that a view based on attribute values is more natural. In OASIS, we use the uniform abstraction of *environmental predicate* for testing parameters of both of these kinds.

In GRBAC, context is checked only on object access, whereas in OASIS environmental predicates can be included in role activation rules as well as in authorization rules. GRBAC is potentially more expensive to implement than OASIS because a GRBAC system must keep track of all relevant active object and environment roles using its Role Activation Service. In OASIS, role activation is potentially conditional on the context, which allows tests to be performed once per session rather than at each access. OASIS was designed to be scalable from the start, and only minimal state has to be maintained in our implementation based on active platforms. The validity of environmental predicates in membership rules is ensured through active notification, which further reduces the unnecessary use of resources. We believe that the real test of architectures such as GRBAC and OASIS will come through implementing substantial applications in which both policy specification and services are distributed; we have done our best to ensure that our design will prove both scalable and manageable.

8. RELATING THE IMPLEMENTATION TO THE MODEL

We have worked on OASIS for some time and have presented its architecture, design and distributed system engineering issues in Hayton et al. [1998], Bacon et al. [2000, 2001b], Hine et al. [2000], and Bacon and Moody [2002]. Wide area, multidomain applications are becoming increasingly common, and there is a need to federate policy as expressed by administrators in cooperating, yet autonomous domains. A coherent, logical model is essential in order to achieve this, but a model is useful only if it represents the semantics of the access control system faithfully. We have developed the model in part because we recognized that it was important to base the implementation on a clear formal specification, and we are at present using the model as a reference while creating tools to manage access control policy.

In systems where confidential data is transferred across networks security is a crucial requirement; that is, secure communication, authentication, access control and audit. A great deal of work has been carried out on some of these aspects of security and it is possible to build a secure communications infrastructure, with authenticated principals, from standard products. We have integrated RBAC with a public key infrastructure (PKI) [Bacon et al. 2001b].

One of the aims of OASIS that is addressed in the formal model presented here is to provide active support for context-sensitive access control. Both role activation and authorization rules can include *environmental predicates*, which test aspects of the context in which the rule is validated. In the case of activation rules, an environmental predicate can be declared a *membership condition*, which must continue to hold while the role is active. Requests for service invocation must satisfy the service's authorization policy, which will require the presentation of an appropriate role membership certificate (RMC). At each

invocation, the RMC must be validated by the issuing service, which has to check in particular that none of the membership conditions has been violated. Checking these conditions may involve other services, possibly in external domains. To avoid external validation on every invocation, RMCs may be cached by the authorizing service between invocations. If the certificate becomes invalid, for example because some membership condition is violated, then the service that issued the RMC notifies the authorizing service. Distributed systems are subject to partial failures and partitions. If an external validator cannot be contacted, or an event channel to it ceases to deliver heartbeats, then the service must follow its policy on whether to allow invocations that are affected to proceed. This policy will reflect the service's chosen trade-off between paranoia and the desire to support continuing availability.

Designing an architecture, specifying a model and building an implementation are only part of the story. The real test of the OASIS system will come when it can be evaluated in a live application. We are continuing to interact with the Eastern Region Electronic Health Record Consortium (EREHRC), a group that includes a number of senior managers in the Eastern Region of the UK National Health Service. EREHRC is hoping to develop a prototype system for managing EHRs in the Eastern Region, and we shall experiment with the use of OASIS to provide access control. It is essential that there are tools to manage, configure and deploy access control policy so that domain administrators can be sure that our software enforces their intentions.

We have built an extension to the PostgreSQL Object-Relational Database Management System that supports active predicates, which we are using both for environmental predicates and for active policy management. The result of an environmental predicate that depends on database lookup can be cached at the authorizing service, which is notified if the predicate becomes false. Each administrative domain has a policy store that holds metadata describing the access control structure of each service managed by OASIS: role names, appointments, privileges, environmental predicates and the sets of rules that cover both role activation and authorization. We are at present designing the naming scheme that will bind the entities of the formal model into the runtime environment.

9. CONCLUSIONS AND FUTURE WORK

OASIS is an access control system for open, interworking services in a distributed environment. Services may be developed independently, as part of a loose federation of administrative domains, but service level agreements (SLAs) allow their secure interoperation. OASIS is closely integrated with an active, event-based middleware infrastructure. In this way, we continuously monitor applications within their environment, ensuring that security policy is satisfied at all times. We therefore address the needs of distributed applications that require active security. Any formalization must take account of the relationship between OASIS and the underlying active platform.

In this article, we have formalized the OASIS model. OASIS is role-based: services name their client roles and enforce policy for role activation and service

invocation, expressed in terms of their own and other services' roles. A signed role membership certificate is returned to the user on successful role activation and this may be used as a credential for activating other roles, according to policy.

We do not use role delegation. Instead, we have defined *appointment*, which we believe to be both more intuitive and more applicable in practice. Appointments may be long-lived, such as academic and professional qualifications, or transient, such as standing in for a colleague who is called away while on duty. On appointment, the appointee is issued with an appointment certificate that may be used, together with any other credentials required by policy, to activate one or more roles.

In addition to role membership certificates and appointment certificates, role activation rules may include environmental constraints. Examples are user-independent constraints such as time of day and conditions on user-dependent parameters. For example, it may be necessary to perform database lookup at a service to ascertain that the user is a member of some group. Alternatively, a simple parameter check may ascertain that the user is a specified exception to a general category who may activate the role.

The membership rule for a role indicates those security predicates for activating the role that must remain true while the role is active. Event channels are set up between services to monitor membership conditions of the rule. Should any condition become false this triggers an event notification to the role-issuing service and the role is deactivated for that user session. By this means, we maintain an active security environment.

OASIS is session based. Starting from initial roles, such as "authenticated, logged-in user," a user may activate a number of roles by submitting the credentials required to satisfy an activation rule. The activated roles therefore form trees dependent on initial roles. Should any membership condition for any role become false the dependent subtree is collapsed. If a single initial role is deactivated (the user logs out), all the active roles collapse and the session terminates.

Our application domains require parametrization of roles. For example, in the healthcare domain a patient might specify "all doctors except my uncle Fred Smith may read my health record". For a filing system, it is necessary to indicate individual owners of files as well as groups of users. In this article, we have extended Yao et al. [2001] with the detailed modelling of role parameters.

In practice, distributed systems comprise many domains; for example, the healthcare domain comprises hospitals, primary care practices, research institutes, etc. We will generalize our naming structure to include domains explicitly. We are working on the management of policy for role activation and service use. Policy may derive from local and national sources and will change over time. Policy stores will be managed using OASIS in our active environment. The formalization of OASIS will provide a firm basis for requirements such as checking the consistency of policies.

Formal specification is crucial in order to manage access control policy for future, large scale, widely distributed, multidomain systems. A formal model allows policy components established across a number of domains to be checked

for consistency. This is necessary, since, otherwise, policy cannot be deployed by domains acting autonomously; for example, a government edict might require changes of policy across heterogeneous healthcare domains. Automation is essential to minimise human error, and it can only be used safely when there is a formal model. We are experimenting with the use of metapolicies that will enable SLAs between domains to be developed automatically [Belokosztolszki and Moody 2002].

A major advantage of a formal model is that it becomes possible to reason about the relationship between a specified policy and its implementation. If role activation and authorisation rules are correctly implemented then the specified policy will be enforced. There are two particular difficulties in verifying the correctness of an application protected by OASIS: first, OASIS security is *active*, and it is necessary to reason about the handling of membership conditions; second, in addition to checks on access control entities, rules can include environmental predicates, whose evaluation can in principle involve an arbitrary computation, with possible side effects on the access control regime itself. We intend to reason about the state of an application's access control system by considering snapshots of state taken between occurrences of role activation and role deactivation. Explicit deactivation occurs when a role issuing service is notified that a membership condition no longer holds. If the heartbeat to an external service responsible for a membership condition is lost, then the issuing service must decide what action to take. Any formal proof of correctness will be relative both to the properties of the code that evaluates environmental predicates and to the implementation of the active platform.

ACKNOWLEDGMENTS

We acknowledge the support of the Engineering and Physical Sciences Research Council (EPSRC) under the grant *OASIS Access Control: Implementation and Evaluation* (GR/M75686/01). Members of the Opera research group in the Computer Laboratory made helpful comments on earlier drafts of this article. We are grateful to Jon Tidswell whose guidance improved [Yao et al. 2001] considerably, and to Trent Jaeger and Ravi Sandhu for many helpful comments during the preparation of the final version. The ideas that lie behind Definition 5.8 were introduced by John Hine [Hine et al. 2000].

REFERENCES

AHN, G.-J. AND SANDHU, R. S. 2000. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Sec. 3*, 4 (Nov.), 207–226.

BACON, J. M., LLOYD, M., AND MOODY, K. 2001a. Translating role-based access control policy within context. In *Policy 2001, Workshop on Policies for Distributed Systems and Networks*. Lecture Notes in Computer Science, vol. 1995. Springer-Verlag, Heidelberg and New York, 107–119.

BACON, J. M. AND MOODY, K. 2002. Toward open, secure, widely distributed services. *Commun. ACM 45*, 6 (June), 49–53.

BACON, J. M., MOODY, K., BATES, J., HAYTON, R. J., MA, C., MCNEIL, A., SEIDEL, O., AND SPITERI, M. D. 2000. Generic support for distributed applications. *IEEE Comput. 33*, 3 (Mar.), 68–76.

BACON, J. M., MOODY, K., AND YAO, W. T. M. 2001b. Access control and trust in the use of widely distributed services. In *Middleware 2001*. Lecture Notes in Computer Science, vol. 2218. Springer-Verlag, Heidelberg and New York, 300–315.

BARKA, E. AND SANDHU, R. S. 2000a. A role-based delegation model and some extensions. In *Proceedings of the 23rd National Information Systems Security Conference* (NISSC 2000) (Baltimore, Md., Oct. 16–19). See http://csrc.nist.gov/nissc/2000/proceedings/toc.pdf.

BARKA, E. AND SANDHU, R. S. 2000b. Framework for role-based delegation models. In *Proceedings of the 16th Annual Computer Security Applications Conference* (ACSAC 2000) (New Orleans, La. Dec. 11–15). IEEE Computer Society Press, Los Alamitos, Calif., 168–177.

BELOKOSZTOLSZKI, A. AND MOODY, K. 2002. Meta-policies for distributed role-based access control systems. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks* (Policy 2002) (Monterey, Calif., June 5–7). IEEE Computer Society Press, Los Alamitos, Calif., 106–115.

BERTINO, E., BETTINI, C., FERRARI, E., AND SAMARATI, P. 1996. A temporal access control mechanism for database systems. *IEEE Trans. Knowl. Eng. 8*, 1 (Feb.), 67–80.

BERTINO, E., BONATTI, P. A., AND FERRARI, E. 2000. TRBAC: A temporal role-based access control model. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control* (RBAC '00) (Berlin, Germany, July 26–27). ACM, New York, 21–30.

BERTINO, E., FERRARI, E., AND ATLURI, V. 1997. A flexible model for the specification and enforcement of role-based authorizations in workflow management systems. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (RBAC '97) (Fairfax, Va., Nov. 6–7). ACM, New York, 1–12.

BIRON, P. AND MALHOTRA, A. 2001. *XML schema part 2: Datatypes*. World Wide Web Consortium (W3C) recommendation 02 May 2001. Available at http://www.w3.org/TR/xmlschema-2/.

BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSOHN, N., NIELSEN, H. F., THATTE, S., AND WINER, D. 2000. *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web Consortium (W3C) note 08 May 2000. Available at http://www.w3.org/TR/SOAP/.

COVINGTON, M. J., LONG, W., SRINIVASAN, S., DEY, A., AHAMAD, M., AND ABOWD, G. 2001. Securing context-aware applications using environment roles. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies* (SACMAT 2001) (Chantilly, Va., May 3–4). ACM, New York, 10–20.

COVINGTON, M. J., MOYER, M. J., AND AHAMAD, M. 2000. Generalized role-based access control for securing future applications. In *Proceedings of the 23rd National Information Systems Security Conference*. (NISSC 2000) (Baltimore, Md., Oct. 16–19). See http://csrc.nist.gov/nissc/2000/proceedings/toc.pdf.

CRISPO, B. 1998. Delegation of responsibility. In *Proceedings of the 6th International Security Protocols Workshop* (Cambridge, UK, Apr. 15–17). Lecture Notes in Computer Science, vol. 1550. Springer-Verlag, Heidelberg and New York, 118–130.

FERRAIOLO, D. F., BARKLEY, J. F., AND KUHN, D. R. 1999. A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Sec. 2*, 1 (Feb.), 34–64.

GEORGIADIS, C., MAVRIDIS, I., PANGALOS, G., AND THOMAS, R. K. 2001. Flexible team-based access control using contexts. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies* (SACMAT 2001) (Chantilly, Va., May 3–4). ACM, New York, 21–30.

GIURI, L. AND IGLIO, P. 1997. Role templates for content-based access control. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (RBAC '97) (Fairfax, Va., Nov. 6–7). ACM, New York, 153–159.

GLIGOR, V. D., GAVRILA, S., AND FERRAIOLO, D. 1998. On the formal definition of separation of duty policies and their composition. In *Proceedings of 1998 IEEE Symposium on Security and Privacy* (Oakland, Calif., May 3–6). IEEE Computer Society Press, Los Alamitos, Calif., 172–183.

GONG, L. 1989. A secure identity-based capability system. In *Proceedings of 1989 IEEE Symposium on Security and Privacy* (Oakland, Calif., May). IEEE Computer Society Press, Los Alamitos, Calif., 56–63.

HAYTON, R. J., BACON, J., AND MOODY, K. 1998. OASIS: Access control in an open, distributed environment. In *Proceedings of 1998 IEEE Symposium on Security and Privacy* (Oakland, Calif., May 3–6). IEEE Computer Society Press, Los Alamitos, Calif., 3–14.

HINE, J. H., YAO, W. T. M., BACON, J. M., AND MOODY, K. 2000. An architecture for distributed OASIS services. In *Middleware 2000* (Palisades, N.Y., Apr. 4–8). Lecture Notes in Computer Science, vol. 1795. Springer-Verlag, Heidelberg and New York, 104–120.

JAEGER, T. 1999. On the increasing importance of constraints. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control* (RBAC '99) (Fairfax Va., Oct. 28–29). ACM, New York, 33–42.

KANDALA, S. AND SANDHU, R. S. 2002. Secure role-based workflow models. In *Proceedings of the 15th IFIP WG 11.3 Working Conference on Database Security (Database and Application Security XV: Status and Prospects)* (Niagara on the Lake, Canada, July 15–18, 2001). Kluwer Academic Publishers, Dordrecht, The Netherlands, 45–58.

KUHN, D. R. 1997. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (RBAC '97) (Fairfax, Va., Nov. 6–7). ACM, New York, 23–30.

MONJIAN, B. 2000. *PostgreSQL: Introduction and Concepts*, ISBN: 0-201-70331-9. Addison-Wesley, Boston, Mass.

NYANCHAMA, M. AND OSBORN, S. 1995. Access rights administration in role-based security systems. In *Proceedings of the 8th IFIP WG 11.3 Working Conference on Database Security (Database Security VIII: Status and Prospects)* (Bad Salzdetfurth, Germany, August 23–26). North-Holland (Elsevier), Amsterdam, The Netherlands, 37–56.

NYANCHAMA, M. AND OSBORN, S. 1999. The role graph model and conflict of interest. *ACM Trans. Inf. Syst. Sec. 2*, 1 (Feb.), 3–33.

OH, S. AND SANDHU, R. S. 2002. A model for role administration using organization structure. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies* (SACMAT 2002) (Monterey, Calif., June 3–4). ACM, New York, 155–162.

SALTZER, J. H. AND SCHROEDER, M. D. 1975. The protection of information in computer systems. *Proc. IEEE 63,* 9 (Sept.), 1278–1308.

SANDHU, R. S., BHAMIDIPATI, V., AND MUNAWER, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Sec. 2*, 1 (Feb.), 105–135.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Comput. 29*, 2 (Feb.), 38–47.

SIMON, R. T. AND ZURKO, M. E. 1997. Separation of duty in role-based environments. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations* (Rockport, Mass., June 10–12). IEEE Computer Society Press, Los Alamitos, Calif., 183–194.

THOMAS, R. K. 1997. Team-based access control (TMAC): A primitive for applying role-based access controls in collaborative environments. In *Proceedings of the 2nd ACM Workshop on Role-Based Access Control* (RBAC '97) (Fairfax, Va., Nov. 6–7). ACM, New York, 13–19.

THOMAS, R. K. AND SANDHU, R. S. 1997. Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In *Proceedings of the 11th IFIP WG 11.3 Workshop on Database Security (Database Security XI: Status and Prospects)* (Lake Tahoe, Calif., Aug. 10–13). Chapman and Hall, 166–181.

THOMPSON, H. S., BEECH, D., MALONEY, M., AND MENDELSOHN, N. 2001. *XML Schema Part 1: Structures*. World Wide Web Consortium (W3C) recommendation 02 May 2001. Available at http://www.w3.org/TR/xmlschema-1/.

WANG, W. 1999. Team-and-role-based organizational context and access control for cooperative hypermedia environments. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia* (Hypertext'99). ACM, New York, 37–46.

YAO, W. T. M., MOODY, K., AND BACON, J. M. 2001. A model of OASIS role-based access control and its support for active security. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies* (SACMAT 2001) (Chantilly, Va., May 3–4). ACM, New York, 171–181.

Received October 2001; revised January 2002, February 2002, July 2002; accepted July 2002